Universidad de Murcia

Grado en Física

LABORATORIO DE ORDENADORES

Apuntes de teoría, ejemplos prácticos y
ejercicios resueltos.

Autor: David Conesa Pagán

Curso Académico 2024 / 2025

1.ª Edición (Revisada)

ESTE EJEMPLAR PERTENECE AL ALUMNO/A:

.....

Laboratorio de Ordenadores – Edición Revisada 2025

© David Conesa Pagán

Obra registrada bajo licencia Creative Commons (CC BY-NC-ND 4.0)

Código de Registro: 2602144563045

El uso no autorizado o la reproducción de este material
está protegido por la Ley de Propiedad Intelectual.

Índice

1. Problema 1: Muelle **4**
1.1. Método de la Segunda derivada: . . . 4
1.1.1. Creamos los arrays . . . 6
1.1.2. Función teórica: . . . 6
1.1.3. Algoritmo segunda derivada: . . . 6
1.1.4. Gráficas: . . . 6
1.1.5. Función teórica: . . . 8
1.1.6. Algoritmo segunda derivada velocidad: . . . 8
1.1.7. Gráficas: . . . 8
1.1.8. Funciones: . . . 9
1.1.9. Funciones con el algoritmo: . . . 9
1.1.10. Gráfica: . . . 9
1.2. Algoritmo de Euler: . . . 10
1.2.1. Creamos los arrays . . . 11
1.2.2. Funciones teóricas: . . . 11
1.2.3. Algoritmo de Euler: . . . 11
1.2.4. Gráficas: . . . 11
1.2.5. Funciones teóricas: . . . 13
1.2.6. Funciones con algoritmo: . . . 13
1.2.7. Gráfica: . . . 13
1.3. Algoritmo de Euler modificado: . . . 14
1.3.1. Creamos los arrays: . . . 14
1.3.2. Algoritmo de Euler modificado: . . . 14
1.3.3. Gráficas . . . 14
1.3.4. Algoritmo de euler modificado (energía): . . . 16
1.3.5. Gráfica: . . . 16
1.4. Oscilador anarmónico: . . . 17
1.4.1. Creamos los arrays . . . 18
1.4.2. Algoritmo de euler: . . . 18
1.4.3. Algoritmo de euler modificado: . . . 19
1.4.4. Gráficas: . . . 19

2. Problema 2: Problema de los dos cuerpos **22**
2.1. Algoritmo de Euler: . . . 24
2.1.1. Creamos los arrays: . . . 24
2.1.2. Funciones: . . . 24
2.1.3. Gráficas: . . . 25
2.2. Algoritmo de Euler modificado: . . . 25
2.2.1. Creamos los arrays: . . . 26
2.2.2. Funciones: . . . 26
2.2.3. Gráficas: . . . 26
2.3. Periodo: . . . 27
2.4. Conservación del momento angular: . . . 27
2.5. Conservación de la energía: . . . 28
2.5.1. Creamos los arrays: . . . 28
2.5.2. Funciones: . . . 28
2.6. Coordenadas en el perihelio, longitud semiejes y excentricidad: . . . 30

3. Extra: Péndulo **32**

4. Extra: Muelle en dos dimensiones **38**

1. Problema 1: Muelle

Para realizar esta práctica, es obvio que partimos desde el conocimiento de la fórmula que rige la fuerza de un muelle:

$$\vec{F} = -k\vec{x} = -kx \cdot \vec{e}_r$$

Con la cual eramos capaces de describir el movimiento armónico simple de un muele (M.A.S.), de la forma que la igualábamos con la conocida segunda ley de Newton $\vec{F} = m\vec{a}$

Para este ejercicio se presenta que tenemos de incógnita $x(t)$, la cual es una función de variable real, es decir $x(t) : \mathbb{R} \to \mathbb{R}$, pero la tendremos que hallar en función de una ecuación diferencial al ser esta una incógnita, por tanto:

$$ma = -kx \Longrightarrow m\frac{d^2x(t)}{dt^2} = -kx(t) \Longrightarrow m\frac{d^2x(t)}{dt^2} + kx(t) = 0$$

La cual también podemos expresar como $mx''(t) + kx(t) = 0$

Para resolver este problema tenemos que asignar los siguientes puntos:

- $x(t = 0) = x_0$
- $v(t = 0) = x'(t = 0) = v_0 = 0$

Por lo que la pregunta sería: ¿$\exists x(t) : \mathbb{R} \to \mathbb{R} / mx''(t) + kx(t) = 0 \ \forall t$?

1.1. Método de la Segunda derivada:

Para dar la respuesta tenemos que recurrir al cálculo numérico. Lo primero es pasar de un tiempo continuo, a un tiempo discreto, esto es una aproximación que obviamente nos llevará a error, pero ese es el precio a seguir, concretamente:

$$0 = t_0 < t_1 < t_2 < ... < t_i < ... < t_N = t_f$$

Por tanto si tenemos un tiempo inicial t_0 y un tiempo final t_f, lo más cómodo sería dividirlo en N pasos, es decir:

$$\Delta t = \frac{t_f - t_0}{N}$$

De manera que, para un instante determinado de tiempo t_i, obtenemos la expresión:

$$t_i = t_0 + i\Delta t$$

Ahora necesitamos sacar una expresión para $x''(t_i)$, que para ello usaremos las series de Taylor:

- Para $x(t_i + \Delta t)$

$$x(t_i + \Delta t) = x(t_i) + x'(t_i)\Delta t + \frac{1}{2}x''(t_i)\Delta t^2 + ...$$

- Para $x(t_i - \Delta t)$

$$x(t_i - \Delta t) = x(t_i) - x'(t_i)\Delta t + \frac{1}{2}x''(t_i)\Delta t^2 - ...$$

Procedemos a sumarlas:

$$x(t_i + \Delta t) + x(t_i - \Delta t) = 2x(t_i) + 2\frac{1}{2}x''(t_i)\Delta t^2$$

$$x(t_i + \Delta t) + x(t_i - \Delta t) = 2x(t_i) + x''(t_i)\Delta t^2$$

Y ahora procedemos a despejar $x''(t_i)$:

$$x''(t_i)\Delta t^2 = x(t_i + \Delta t) + x(t_i - \Delta t) - 2x(t_i) \Longrightarrow x''(t_i) = \frac{x(t_i + \Delta t) + x(t_i - \Delta t) - 2x(t_i)}{\Delta t^2}$$

$$x''(t_i) = \frac{x(t_i + \Delta t) - 2x(t_i) + x(t_i - \Delta t)}{\Delta t^2}$$

Ahora como el tiempo esta discretizado, podemos aproximarlo de la siguiente manera:

$$x''(t_i) \approx \frac{x(t_{i+1}) - 2x(t_i) + x(t_{i-1})}{\Delta t^2}$$

Ahora, una vez que tenemos bien definido nuestro $x''(t_i)$, podemos sacar la relación:

$$x''(t_i) = -\frac{k}{m}x(t_i)$$

$$\frac{x(t_{i+1}) - 2x(t_i) + x(t_{i-1})}{\Delta t^2} = -\frac{k}{m}x(t_i)$$

$$\frac{x_{i+1} - 2x_i + x_{i-1}}{\Delta t^2} = -\frac{k}{m}x_i$$

Donde $i = 1, 2, 3, ..., N-1$, porque tenemos un x_{i+1} y cuando $i = N-1$ tendremos x_N.

Cabe recalcar que se podría haber hecho haciendo uso de la definición de derivada.

Ahora vamos a despejar x_{i+1}:

$$x_{i+1} - 2x_i + x_{i-1} = -\frac{k}{m}x_i\Delta t^2$$

$$x_{i+1} = 2x_i - x_{i-1} - \frac{k}{m}x_i\Delta t^2$$

$$x_{i+1} = \left(2 - \frac{k}{m}\Delta t^2\right)x_i - x_{i-1}$$

Y como sabemos que $x^{'}(t_0) = v_0 = \lim_{h\to 0}\frac{x(t_0+h) - x(t_0)}{h}$, vemos que entonces $x_1 = x_0 + v_0\Delta t$

$$x(t_1) = x(t_0) + x^{'}(t_0)\Delta t + \frac{1}{2!}x^{''}(t_0)\Delta t^2 ... = x_0 + v_0\Delta t - \frac{1}{2}\frac{k}{m}x_0\Delta t^2$$

Llegando como solución:

$$x_1 = x_0 + v_0\Delta t - \frac{1}{2}\frac{k}{m}\Delta t^2 x_0$$

$$x_1 = \left(1 - \frac{1}{2}\frac{k}{m}\Delta t^2\right)x_0 + v_0\Delta t$$

Por tanto finalizamos con las siguientes datos:

- x_0
- v_0
- $x_1 = \left(1 - \frac{1}{2}\frac{k}{m}\Delta t^2\right)x_0 + v_0\Delta t$
- $x_{i+1} = \left(2 - \frac{k}{m}\Delta t^2\right)x_i - x_{i-1}$

Ahora vamos a realizar un código para verse el resultado con los siguientes parámetros: Para los valores: $x_0 = 0{,}1m$; $v_0 = 0m/s$; $t_0 = 0s$; $t_f = 150s$; $k = 10^{-4}N/m$; $m = 0{,}01kg$ y tomando intervalos de tiempo de $dt = 1s$.

```
import numpy as np
import matplotlib.pyplot as plt

#Parámetros iniciales
x0=0.1                  #posición inicial
v0=0                    #velocidad inicial
t0=0                    #tiempo inicial
tf=150                  #tiempo final
k=1e-4                  #constante elástica
m=0.01                  #masa
N=150                   #Intervalos
dt=int((tf-t0)/(N))     #Delta t
w=(k/m)**(1/2)          # Velocidad angular
```

1.1.1. Creamos los arrays

Importante poner que los divida en N+1 intervalos porque en realidad hacemos N divisiones, pero hay N+1 trozos, por ejemplo si hacemos dos divisiones tenemos 3 trozos.

```
t=np.linspace(t0,tf,N+1)
x=np.zeros(N+1)
v=np.zeros(N+1)
```

1.1.2. Función teórica:

```
def f(x):
    return x0*np.cos(w*t)
```

1.1.3. Algoritmo segunda derivada:

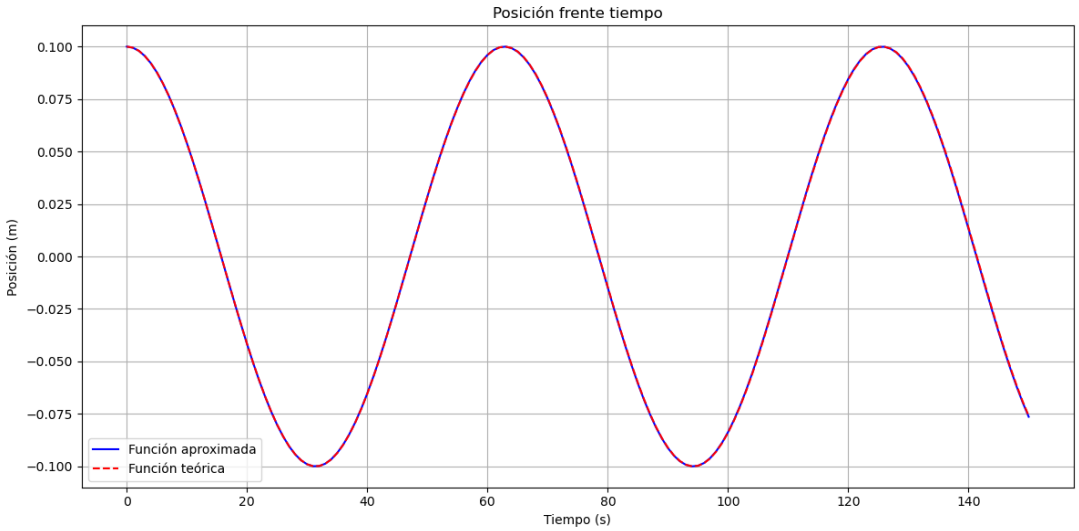
```
v[0]=v0
x[0]=x0
```

```
x[1]=x0+v0*dt-(1/2)*(k/m)*(dt**2)*x0
```

```
for i in range(1,N):    # Ponemos N porque python lo reconoce como N-1
    x[i+1]=(2-(k/m)*dt**2)*x[i]-x[i-1]
```

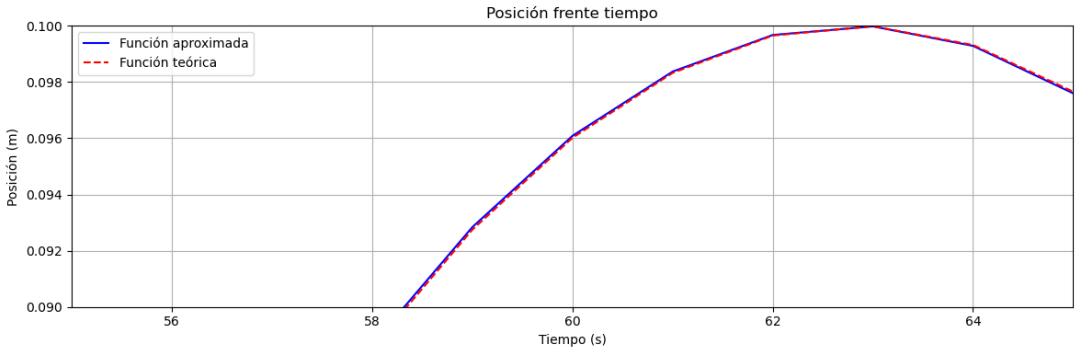
1.1.4. Gráficas:

```
plt.figure(figsize=(12,6))
plt.plot(t,x,label='Función aproximada',color='blue')
plt.plot(t,f(x),label='Función teórica',color='red',linestyle='dashed')
plt.title("Posición frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Posición (m)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```



Si queremos una vista más detallada para ver el error no tenemos más que:

```
plt.figure(figsize=(12,4))
plt.plot(t,x,label='Función aproximada',color='blue')
plt.plot(t,f(x),label='Función teórica',color='red',linestyle='dashed')
plt.title("Posición frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Posición (m)")
plt.grid(True)
plt.tight_layout()
plt.legend()
plt.xlim(55,65)
plt.ylim(0.09,0.1)
plt.show()
```



Si vemos las dos no forman una función trigonométrica, y esto es debido a que aunque estemos usando un código para que represente las imágenes correctas del coseno, no estamos usando muchos puntos, ya que cuantos más usemos, es decir, más grande sea nuestra división de intervalos N y por tanto más pequeño nuestro Δt se irán suavizando las líneas y por consecuente mejorando el resultado, de forma que la función aproximada sea muy parecida a la teórica.

Ahora que sabemos como resolver el problema cuando no sabemos $x(t)$, podemos completarlo intetando calcular la velocidad que lleva el muelle en cualquier instante t, entonces:

$$m \cdot x''(t) = -k \cdot x(t)$$

$$v'(t) = x''(t) = \frac{-k \cdot x(t)}{m}$$

Y como sabemos que $v'(t_0) = \lim_{h \to 0} \frac{v(t_0 + h) - v(t_0)}{h}$, vemos que entonces $v_1 = v_0 + v_0' \Delta t$

$$v_1 = v_0 - \frac{k}{m} \cdot x_0 \Delta t$$

$$v_{i+1} = v_i - \frac{k}{m} x_i \Delta t$$

1.1.5. Función teórica:

```
def v_t(x):
    return -x0*w*np.sin(w*t)
```

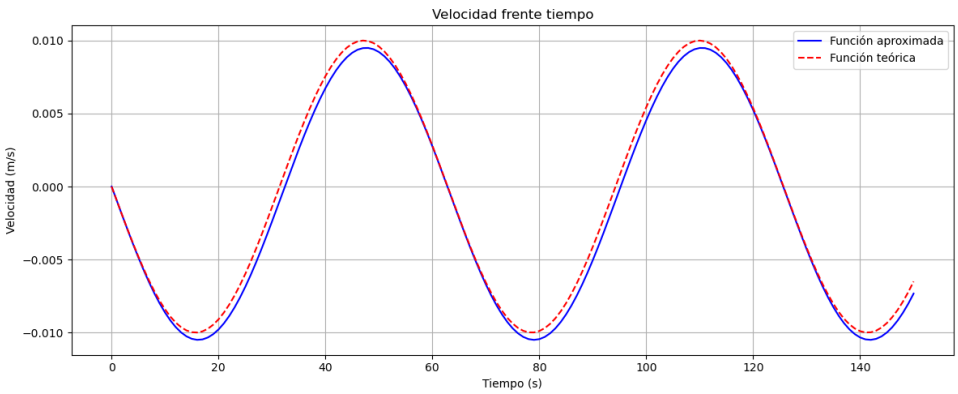
1.1.6. Algoritmo segunda derivada velocidad:

```
v[1] = v0 - (k/m) * x[0] * dt
```

```
for i in range(1,N):
    v[i+1]=v[i]-(k/m)*x[i]*dt
```

1.1.7. Gráficas:

```
plt.figure(figsize=(12,5))
plt.plot(t,v,label='Función aproximada',color='blue')
plt.plot(t,v_t(x),label='Función teórica',color='red',linestyle='dashed')
plt.title("Velocidad frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Velocidad (m/s)")
plt.grid(True)
plt.tight_layout()
plt.legend()
plt.show()
```



Para calcular el periodo podemos hacerlo de la siguiente forma:

```
epsx=0.001
epsv=0.001
for i in range (1,N+1):
    if abs(x[i]-x[0])<epsx and abs(v[i]-v[0])<epsv:
        T=t[i]
        break
print(f" T teórico = {2*np.pi / w} seg")
print (f" T aproximado = {T:.3f} seg")
```

```
 T teórico = 62.83185307179586 seg
 T aproximado = 62.000 seg
```

Hay que decir que esto se basa más en tantear que en otra cosa, ya que nuestros t únicamente toman valores discretos, por tanto nunca llegará a ser el original, en nuestro caso $t = 1, 2, 3, ..,150s$, y a la hora de dar el valor del ε, hay que saber antes cuanto se pueden llegar a diferenciar, por eso es la parte de sacar el periodo es la más tediosa.

Y para finalizar este algoritmo podemos calcular su energía cinética, potencial y mecánica:

1.1.8. Funciones:

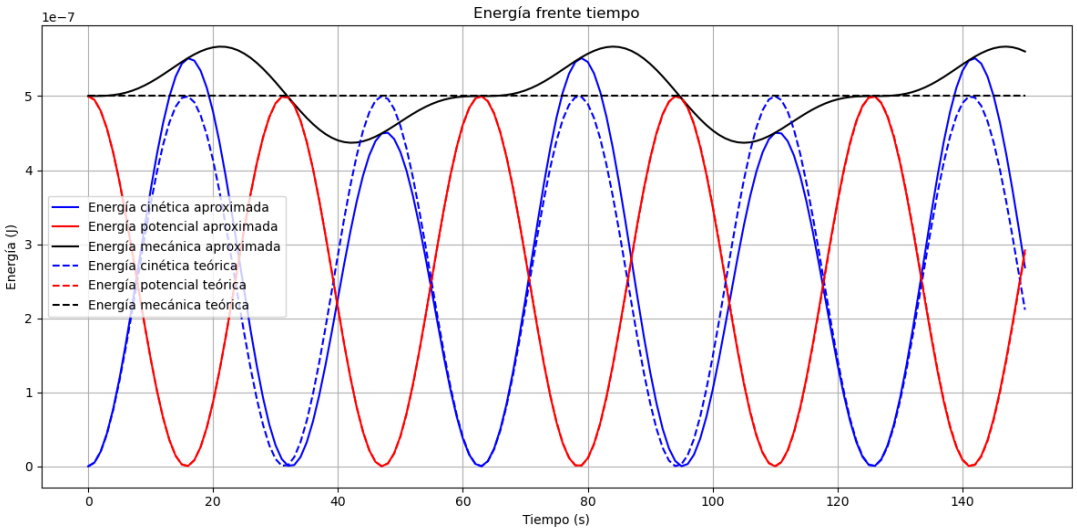
```
def Ek_t(t):
    return 0.5 * m * (-x0*w*np.sin(w*t))**2
def Eu_t(t):
    return 0.5*k*(x0*np.cos(w*t))**2
def Em_t(t):
    return 0.5 * m * (-x0*w*np.sin(w*t))**2 + 0.5*k*(x0*np.cos(w*t))**2
```

1.1.9. Funciones con el algoritmo:

```
Ek=[]; Eu=[]; Em=[]
for i in range (0,N+1):
    Ek.append(0.5 * m * v[i]**2)
    Eu.append(0.5 * k * x[i]**2)
    Em.append(Ek[i] + Eu[i])
```

1.1.10. Gráfica:

```
plt.figure(figsize=(12,6))
plt.plot(t,Ek,label='Energía cinética aproximada',color='blue')
plt.plot(t,Eu,label='Energía potencial aproximada',color='red')
plt.plot(t,Em,label='Energía mecánica aproximada',color='black')
plt.plot(t,Ek_t(t),label='Energía cinética teórica',color='blue',linestyle='dashed')
plt.plot(t,Eu_t(t),label='Energía potencial teórica',color='red',linestyle='dashed')
plt.plot(t,Em_t(t),label='Energía mecánica␣
 ↪teórica',color='black',linestyle='dashed')
plt.title("Energía frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Energía (J)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```



Como vemos, no es muy preciso al final, al revisar la energía vemos que esta no se conserva debidamente, y el fallo lo encontramos en la velocidad, ya que el algoritmo que utiliza no es tan preciso como el de la posición.

Aunque este código tiene buenas aproximaciones, de ser usado en condiciones evaluables para Chicón estaríamos archisuspensos porque es un código muy "chipirritifláutico". Por tanto, los que tenemos que usar de cara al desarrollo de la asignatura y de la evaluación son los algoritmos posteriores.

1.2. Algoritmo de Euler:

Aunque el algoritmo de la segunda derivada es bueno, existen muchos más e incluso mejores, en este algoritmo, vamos a calcular $x(t)$ pero sin conocer la expresión que hallamos antes de $x''(t) = \lim_{\Delta t \to 0} \dfrac{x(t+\Delta t) - 2x(t) + x(t-\Delta t)}{\Delta t^2}$. La diferencia es que en este algoritmo usaremos la fuerza:

$$F = ma \iff a = \frac{F(x(t))}{m} \iff \frac{d^2x(t)}{dt^2} = \frac{F(x(t))}{m}$$

Donde de aquí sacamos las siguientes relaciones:

$$\frac{dx}{dt} = v(t)$$
$$\frac{dv}{dt} = \frac{1}{m}F(x(t))$$

Ahora se ha pasado de un ejercicio con una única variable a un sistema de dos ecuaciones con dos incógnitas y, a priori, más complicado, sin embargo, ambas ecuaciones del sistema están sometidas a una derivada de primer orden para la que sí se conoce aproximación.

Aplicando lo propio del cálculo numérico, la renuncia de la continuidad y, por tanto, de la exactitud en el cálculo de límites, el algoritmo quedaría de la siguiente forma:

$$x'(t_i) \approx \frac{x_{i+1} - x_i}{\Delta t} = v_i$$
$$v'(t_i) \approx \frac{v_{i+1} - v_i}{\Delta t} = \frac{1}{m}F(x_i)$$

Que haciendo unos despejes sencillos conseguimos el algoritmo de Euler:

$$x_{i+1} = x_i + x'(t_i)\Delta t \Longrightarrow x_{i+1} = x_i + v_i\Delta t$$

$$v_{i+1} = \frac{1}{m}F(x_i)\Delta t + v_i \Longrightarrow v_{i+1} = v_i + \frac{F(x_i)}{m}\Delta t$$

```
import numpy as np
import matplotlib.pyplot as plt

#Parámetros iniciales
x0=0.1                  #posición inicial
v0=0                    #velocidad inicial
t0=0                    #tiempo inicial
tf=150                  #tiempo final
k=1e-4                  #constante elástica
m=0.01                  #masa
N=150                   #Intervalos
dt=int((tf-t0)/(N))     #Delta t
w=(k/m)**(1/2)          # Velocidad angular
```

1.2.1. Creamos los arrays

```
t=np.linspace(t0,tf,N+1)
x_euler=np.zeros(N+1)
v_euler=np.zeros(N+1)
```

1.2.2. Funciones teóricas:

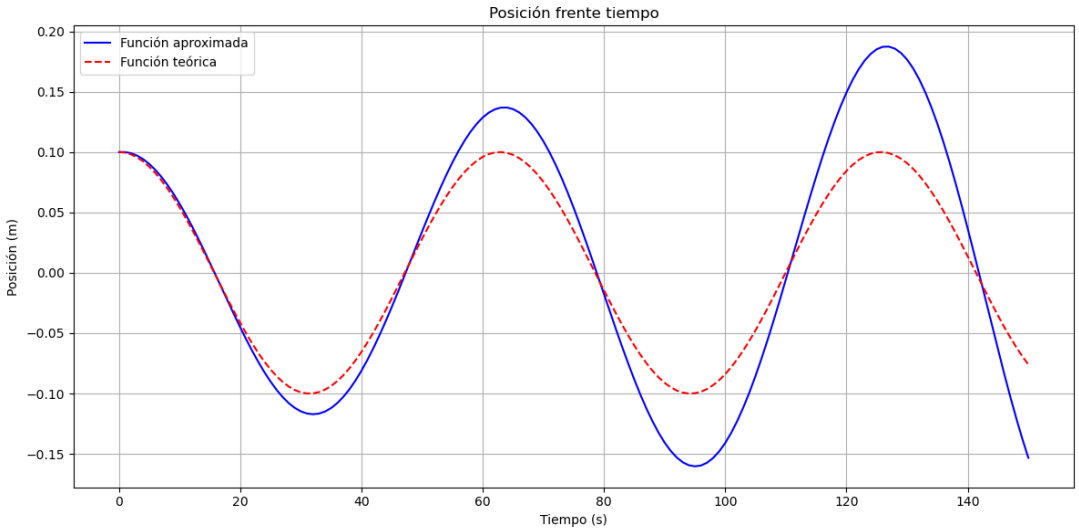
```
def f(x):
    return x0*np.cos(w*t)
def v_t(x):
    return -x0*w*np.sin(w*t)
```

1.2.3. Algoritmo de Euler:

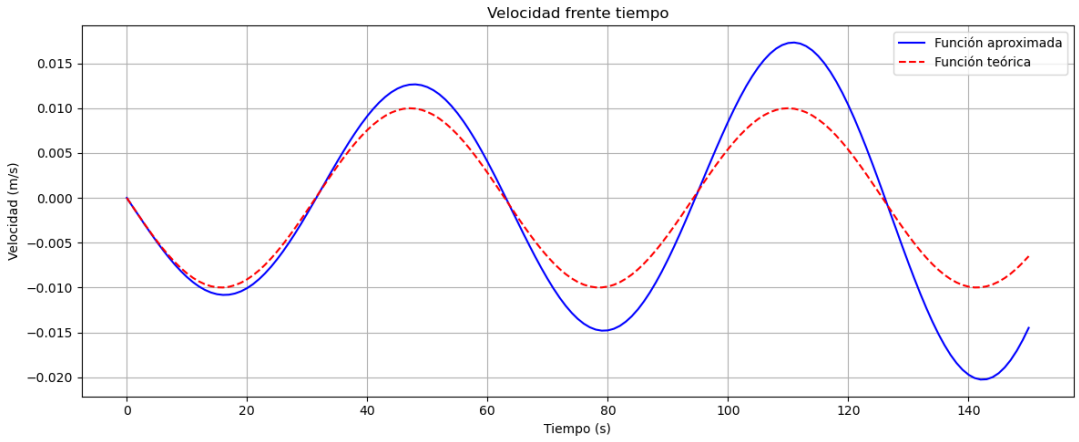
```
x_euler[0]=x0
v_euler[0]=v0

for i in range (0,N):
    x_euler[i+1]=x_euler[i]+v_euler[i]*dt
    v_euler[i+1]=v_euler[i]+((-k*x_euler[i])/m)*dt
```

1.2.4. Gráficas:

```
plt.figure(figsize=(12,6))
plt.plot(t,x_euler,label='Función aproximada',color='blue')
plt.plot(t,f(x),label='Función teórica',color='red',linestyle='dashed')
plt.title("Posición frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Posición (m)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```




```
plt.figure(figsize=(12,5))
plt.plot(t,v_euler,label='Función aproximada',color='blue')
plt.plot(t,v_t(x),label='Función teórica',color='red',linestyle='dashed')
plt.title("Velocidad frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Velocidad (m/s)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```




```
epsx=0.001
epsv=0.001
for i in range (1,N+1):
    if abs(x_euler[i]-x_euler[0])<epsx and abs(v_euler[i]-v_euler[0])<epsv:
        T=t[i]
        break
print(f" T teórico = {2*np.pi / w} seg")
print (f" T aproximado = {T:.3f} seg")
```

```
T teórico = 62.83185307179586 seg
T aproximado = 62.000 seg
```

Y para concluir podemos graficar la energía, aunque ya podemos hacernos una idea de como saldrá:

1.2.5. Funciones teóricas:

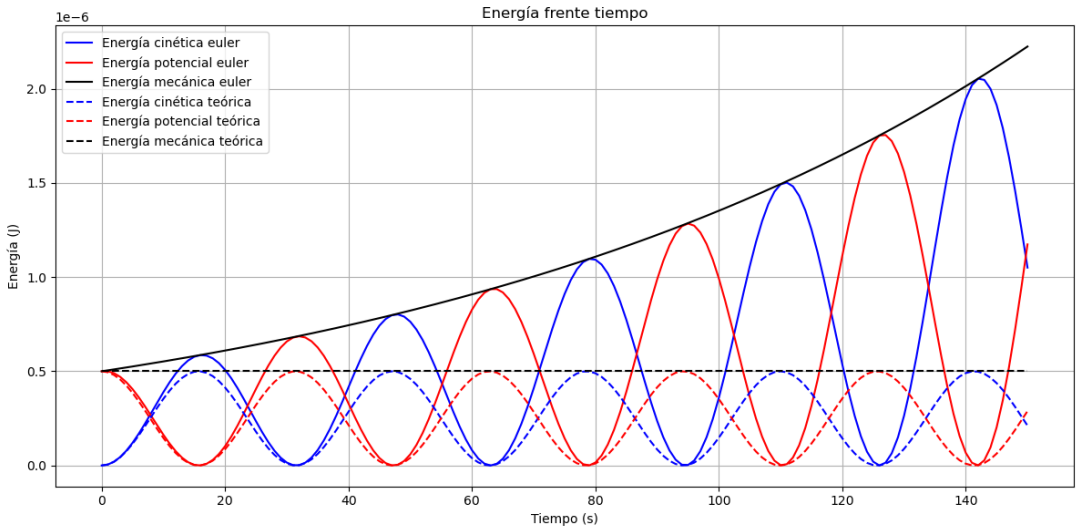
```
def Ek_t(t):
    return 0.5 * m * (-x0*w*np.sin(w*t))**2
def Eu_t(t):
    return 0.5*k*(x0*np.cos(w*t))**2
def Em_t(t):
    return 0.5 * m * (-x0*w*np.sin(w*t))**2 + 0.5*k*(x0*np.cos(w*t))**2
```

1.2.6. Funciones con algoritmo:

```
Ek_euler=[]; Eu_euler=[]; Em_euler=[]
for i in range (0,N+1):
    Ek_euler.append(0.5 * m * v_euler[i]**2)
    Eu_euler.append(0.5 * k * x_euler[i]**2)
    Em_euler.append(Ek_euler[i] + Eu_euler[i])
```

1.2.7. Gráfica:

```
plt.figure(figsize=(12,6))
plt.plot(t,Ek_euler,label='Energía cinética euler',color='blue')
plt.plot(t,Eu_euler,label='Energía potencial euler',color='red')
plt.plot(t,Em_euler,label='Energía mecánica euler',color='black')
plt.plot(t,Ek_t(t),label='Energía cinética teórica',color='blue',linestyle='dashed')
plt.plot(t,Eu_t(t),label='Energía potencial teórica',color='red',linestyle='dashed')
plt.plot(t,Em_t(t),label='Energía mecánica␣
 ↪teórica',color='black',linestyle='dashed')
plt.title("Energía frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Energía (J)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```



1.3. Algoritmo de Euler modificado:

Como hemos podido ver, este algoritmo tiene algo de error, que con el tiempo se va acumulando y genera soluciones como que la energía va creciendo en función del tiempo, algo que sabemos que es rotundamente falso. Ahora el algoritmo de Euler modificado se basa en las siguientes ecuaciones:

$$x_{i+1} = x_i + v_{i+1}\Delta t$$
$$v_{i+1} = v_i + \frac{F(x_i)}{m}\Delta t$$

Por lo que ahora procedemos a realizar los mismos cálculos y veremos la siguiente comparación:

1.3.1. Creamos los arrays:

```
t=np.linspace(t0,tf,N+1)
x_eulermod=np.zeros(N+1)
v_eulermod=np.zeros(N+1)
```

Vamos a omitir las funciones teóricas ya que usaremos las calculadas anteriormente.

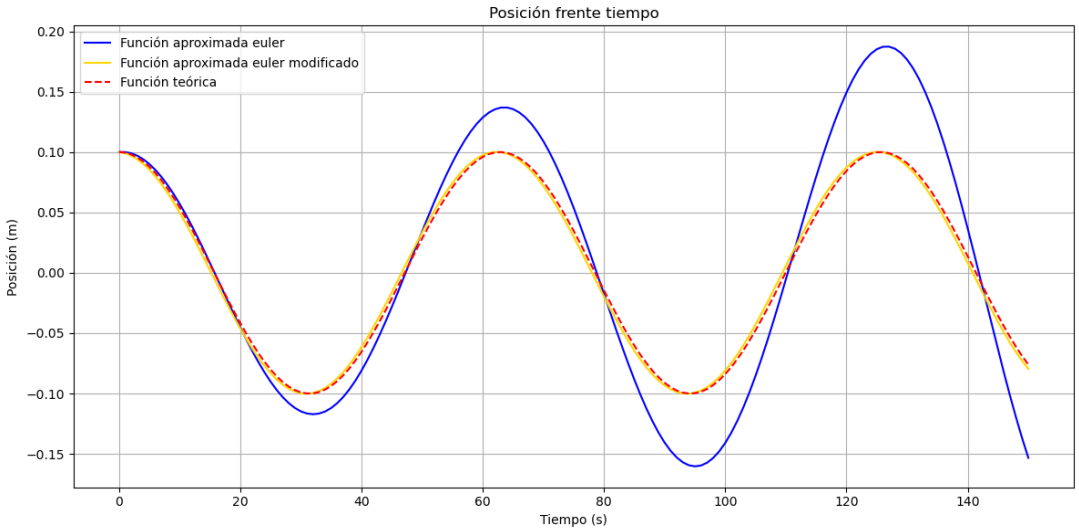
1.3.2. Algoritmo de Euler modificado:

Ahora, cuando hagamos el bucle, tenemos que tener en cuenta que primero debemos poner la velocidad ya que para calcular la posición la necesitará.

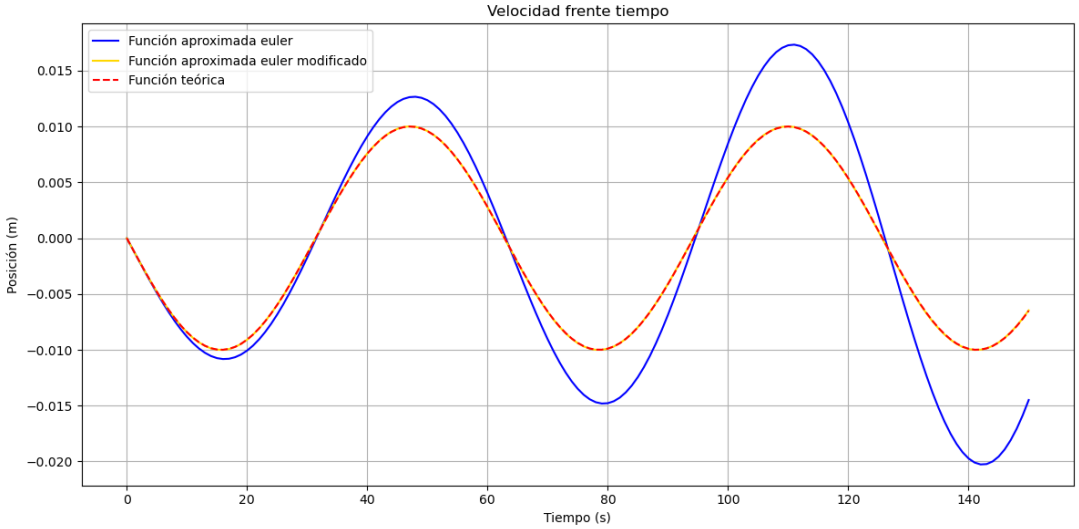
```
x_eulermod[0]=x0
v_eulermod[0]=v0

for i in range (0,N):
    v_eulermod[i+1]=v_eulermod[i]+((-k*x_eulermod[i])/m)*dt
    x_eulermod[i+1]=x_eulermod[i]+v_eulermod[i+1]*dt
```

1.3.3. Gráficas

```
plt.figure(figsize=(12,6))
plt.plot(t,x_euler,label='Función aproximada euler',color='blue')
plt.plot(t,x_eulermod,label='Función aproximada euler modificado',color='gold')
plt.plot(t,f(x),label='Función teórica',color='red',linestyle='dashed')
plt.title("Posición frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Posición (m)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```




```
plt.figure(figsize=(12,6))
plt.plot(t,v_euler,label='Función aproximada euler',color='blue')
plt.plot(t,v_eulermod,label='Función aproximada euler modificado',color='gold')
plt.plot(t,v_t(x),label='Función teórica',color='red',linestyle='dashed')
plt.title("Velocidad frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Posición (m)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```



```
epsx=0.001
epsv=0.001
for i in range (1,N+1):
    if abs(x_eulermod[i]-x_eulermod[0])<epsx and␣
 ↪abs(v_eulermod[i]-v_eulermod[0])<epsv:
        T=t[i]
        break
print(f" T teórico = {2*np.pi / w} seg")
print (f" T aproximado = {T:.3f} seg")
```

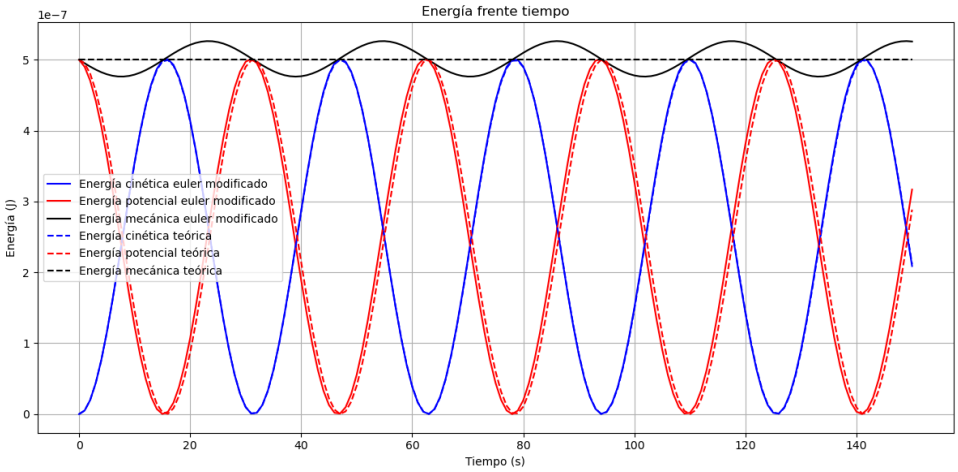
```
T teórico = 62.83185307179586 seg
T aproximado = 62.000 seg
```

1.3.4. Algoritmo de euler modificado (energía):

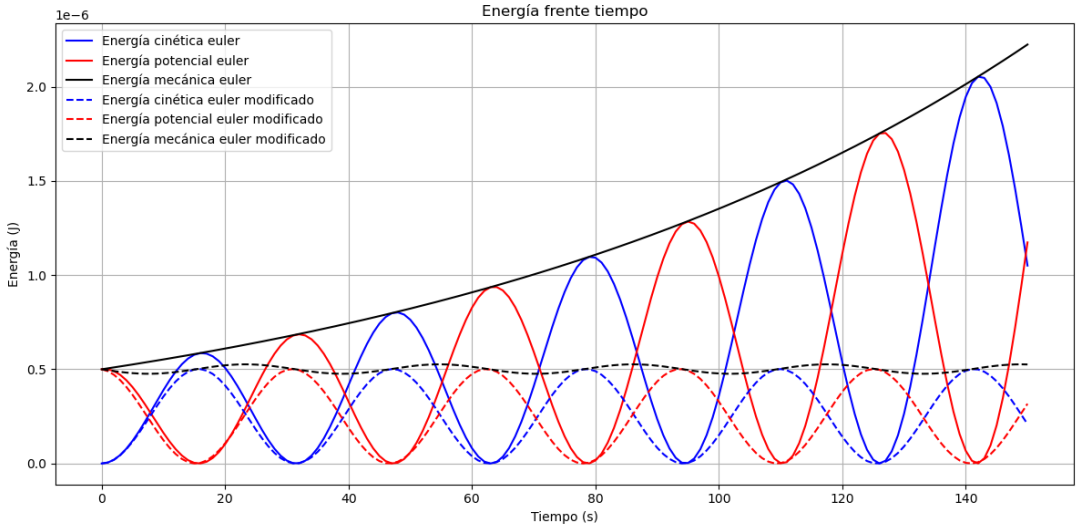
```
Ek_eulermod=[]; Eu_eulermod=[]; Em_eulermod=[]
for i in range (0,N+1):
    Ek_eulermod.append(0.5 * m * v_eulermod[i]**2)
    Eu_eulermod.append(0.5 * k * x_eulermod[i]**2)
    Em_eulermod.append(Ek_eulermod[i] + Eu_eulermod[i])
```

1.3.5. Gráfica:

```
plt.figure(figsize=(12,6))
plt.plot(t,Ek_eulermod,label='Energía cinética euler modificado',color='blue')
plt.plot(t,Eu_eulermod,label='Energía potencial euler modificado',color='red')
plt.plot(t,Em_eulermod,label='Energía mecánica euler modificado',color='black')
plt.plot(t,Ek_t(t),label='Energía cinética teórica',color='blue',linestyle='dashed')
plt.plot(t,Eu_t(t),label='Energía potencial teórica',color='red',linestyle='dashed')
plt.plot(t,Em_t(t),label='Energía mecánica␣
 ↪teórica',color='black',linestyle='dashed')
plt.title("Energía frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Energía (J)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```



```
plt.figure(figsize=(12,6))
plt.plot(t,Ek_euler,label='Energía cinética euler',color='blue')
plt.plot(t,Eu_euler,label='Energía potencial euler',color='red')
plt.plot(t,Em_euler,label='Energía mecánica euler',color='black')
plt.plot(t,Ek_eulermod,label='Energía cinética euler␣
 ↪modificado',color='blue',linestyle='dashed')
plt.plot(t,Eu_eulermod,label='Energía potencial euler␣
 ↪modificado',color='red',linestyle='dashed')
plt.plot(t,Em_eulermod,label='Energía mecánica euler␣
 ↪modificado',color='black',linestyle='dashed')
plt.title("Energía frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Energía (J)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```



Como podemos ver, una imagen dice más que mil palabras, por tanto esto es una prueba más que evidente de que el algoritmo de euler modificado es un método muy sofisticado, que nos permite tanto ahorrar en código como tener una precisión estrepitosa.

1.4. Oscilador anarmónico:

Partimos de que tenemos una expresión para la fuerza, que se rige por $F(x) = -kx^3$, sindo $k > 0$. El problema es que carecemos de $x(t)$, por tanto tenemos que recurrir a los algoritmos mencionados anteriormente.

Como la energía se conserva, tenemos que:

$$U(x) = -\int F(x)dx = k\int x^3 dx = \frac{1}{4}kx^4$$

Entonces, vamos a realizarlo tanto con el algoritmo de euler como el de euler modificado, y vamos a comparar tanto la posición, velocidad y energías de los dos.

- Método de Euler:

$$x_{i+1} = x_i + v_i \Delta t$$
$$v_{i+1} = v_i + \frac{F(x_i)}{m} \Delta t$$

- Método de Euler modificado:

$$x_{i+1} = x_i + v_{i+1} \Delta t$$
$$v_{i+1} = v_i + \frac{F(x_i)}{m} \Delta t$$

```
k=2
m=0.1
x0=0.1
v0=0
w=np.sqrt(k/m)
tf=50
dt=0.1
n=int(tf/dt)
```

1.4.1. Creamos los arrays

```
t3=np.linspace(t0,tf,N+1)
x_euler3=np.zeros(N+1)
v_euler3=np.zeros(N+1)
x_eulermod3=np.zeros(N+1)
v_eulermod3=np.zeros(N+1)
```

```
x_eulermod3[0]=x0
x_euler3[0]=x0
v_eulermod3[0]=v0
v_euler3[0]=v0
```

1.4.2. Algoritmo de euler:

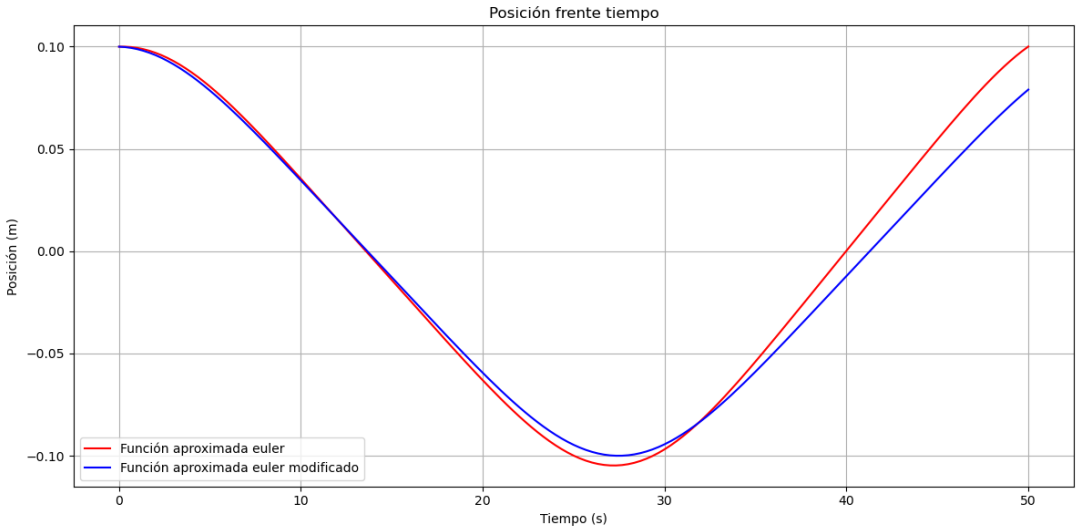
```
for i in range(0,N):
    v_euler3[i+1] = v_euler3[i] + (-k/m) * x_euler3[i]**3 * dt  # Actualización de
 ↪la velocidad
    x_euler3[i+1] = x_euler3[i] + v_euler3[i] * dt  # Actualización de la posición

Ek_euler3=[]; Eu_euler3=[]; Em_euler3=[]
for i in range (0,N+1):
    Eu_euler3.append((1/4) * k * x_euler3[i]**4)
    Ek_euler3.append((1/2)*m* v_euler3[i]**2)
    Em_euler3.append(Ek_euler3[i]+Eu_euler3[i])
```

1.4.3. Algoritmo de euler modificado:

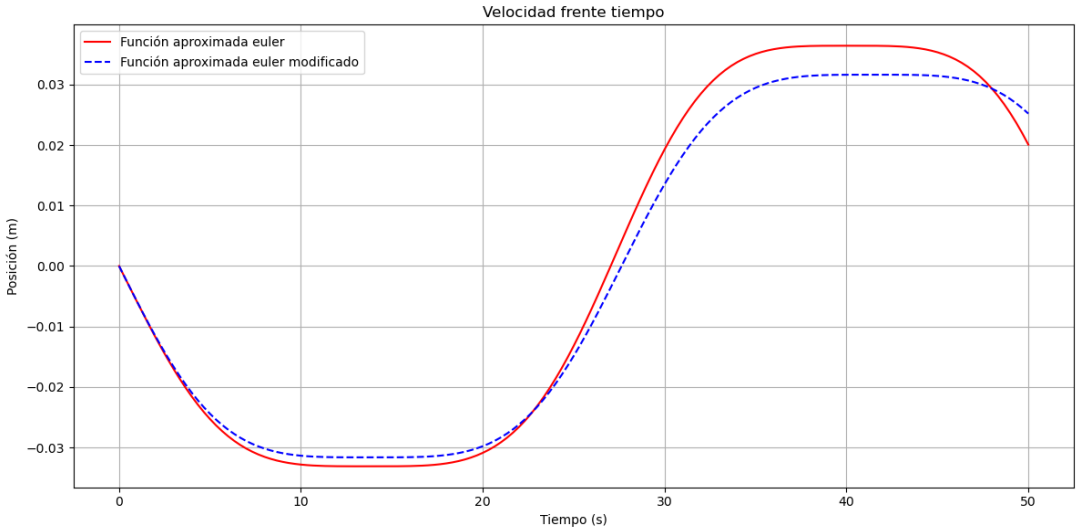
```
for i in range(0,N):
    v_eulermod3[i+1] = v_eulermod3[i] + (-k/m) *x_eulermod3[i]**3 * dt  #␣
 ↪Actualización de la velocidad
    x_eulermod3[i+1] = x_eulermod3[i] + v_eulermod3[i+1] * dt  # Actualización de␣
 ↪la posición

Ek_eulermod3=[]; Eu_eulermod3=[]; Em_eulermod3=[]
for i in range (0,N+1):
    Eu_eulermod3.append((1/4) * k * x_eulermod3[i]**4)
    Ek_eulermod3.append((1/2)*m* v_eulermod3[i]**2)
    Em_eulermod3.append(Ek_eulermod3[i]+Eu_eulermod3[i])
```

1.4.4. Gráficas:

```
plt.figure(figsize=(12,6))
plt.plot(t3,x_euler3,label='Función aproximada euler',color='red')
plt.plot(t3,x_eulermod3,label='Función aproximada euler modificado',color='blue')
plt.title("Posición frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Posición (m)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```




```
plt.figure(figsize=(12,6))
plt.plot(t3,v_euler3,label='Función aproximada euler',color='red')
plt.plot(t3,v_eulermod3,label='Función aproximada euler␣
 ↪modificado',color='blue',linestyle='dashed')
plt.title("Velocidad frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Posición (m)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```



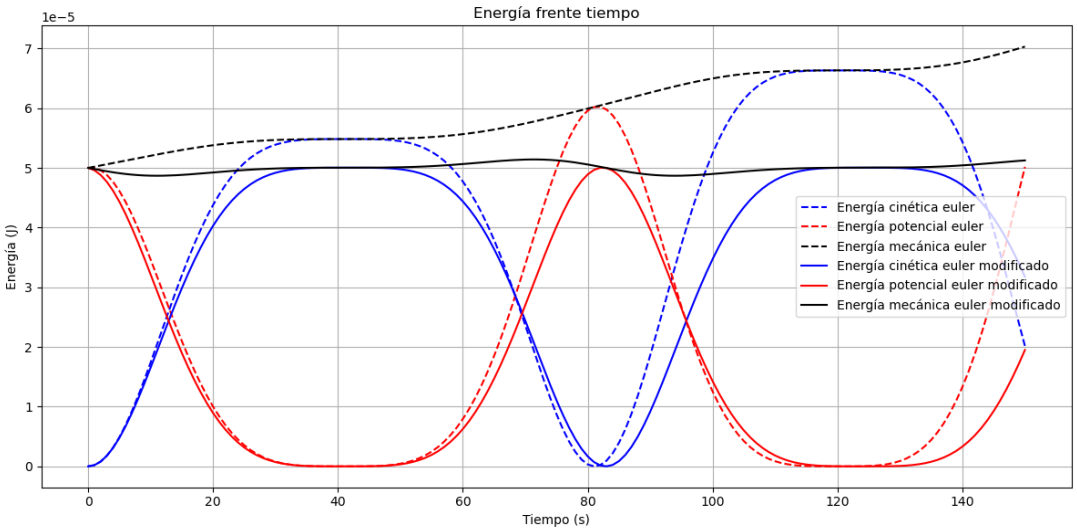
Aqui aunque Chicón nos hizo calcular el periodo, investigando por la red he visto que no podemos calcular el periodo del mismo de la forma en la que estamos acostumbrados, ya que de esa forma, estamos calculado el periodo cuando la fuerza es $F = -kx$, pero en este caso es $F = -kx^3$.

Resolvería este periodo pero nos metemos en un terreno pantanoso para el que todavía no estamos preparados, ya que incluye hasta la *Gamma de Euler*, y como comprenderán, se sale de nuestros alcances, pero si tienen la curiosidad les adjunto la fórmula del periodo teórico:

$$T = \frac{\sqrt{\pi}\Gamma(5/4)}{\Gamma(3/4)}\sqrt{\frac{m}{kA^2}} \approx 2'622\sqrt{\frac{m}{kA^2}}$$

Por lo tanto siento un triste sentimiento de no poder completarlo, por lo que pasaremos a la energía que esta, aunque no la pidió, la veo más importante que el periodo.

```
plt.figure(figsize=(12,6))
plt.plot(t,Ek_euler3,label='Energía cinética euler',color='blue',linestyle='dashed')
plt.plot(t,Eu_euler3,label='Energía potencial euler',color='red',linestyle='dashed')
plt.plot(t,Em_euler3,label='Energía mecánica␣
 ↪euler',color='black',linestyle='dashed')
plt.plot(t,Ek_eulermod3,label='Energía cinética euler modificado',color='blue')
plt.plot(t,Eu_eulermod3,label='Energía potencial euler modificado',color='red')
plt.plot(t,Em_eulermod3,label='Energía mecánica euler modificado',color='black')
plt.title("Energía frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Energía (J)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```



Como podemos ver, *El Algoritmo de Euler Modificado* es mucho, pero mucho más preciso que *El Algoritmo de Euler*.

2. Problema 2: Problema de los dos cuerpos

En este problema, como dice el nombre, se nos plantea un problema en el cual tenemos dos cuerpos que interactúan entre sí, cuya interacción viene dada por la conocida *Ley de Gravitación Universal*. En nuestro casos estos dos cuerpos los conocemos, siento la Tierra y el Sol, es decir, estudiaremos la intección del sistema Tierra-Sol, despreciando cualquier otro astro aunque este se encuentre tanto lo más próximo o lejano posible.

Este problema tiene una solución analítica que podemos calcular con lápiz y papel, pero debemos ser conscientes de que la gran mayoría de problemas no poseen una, de ahí la importancia de conocer métodos numéricos y de cálculo para poder dar una solución consistente. Por lo que llega un punto donde dependemos totalmente de los ordenadores, y saber como usarlos es lo que nos enseña esta asignatura.

Partimos de dos cuerpos con masas m_1 y m_2 respectivamente, con posiciones conocidas en el tiempo 0 de manera que $\vec{r}_1(0)$ y $\vec{r}_2(0)$. Tenemos que hallar la trayectoria que describen ambos cuerpos.

Para la solución de este problema debemos empezar con *La Segunda Ley de Newton*, que llevada a las condiciones del problema tenemos:

$$F(\vec{r},t) = m\vec{a}(t)$$

con la que, haciendo uso de *La Tercera Ley de Newton* obtenemos una ecuación para cada cuerpo, concretamente la fuerza que se ejercen entre ellos:

$$\{\vec{r}_1, \vec{r}_2\}$$

$$m_1\frac{d^2\vec{r}_1(t)}{dt^2} = \vec{F}_{2,1}(\vec{r}_1,\vec{r}_2) \qquad\qquad m_2\frac{d^2\vec{r}_2(t)}{dt^2} = \vec{F}_{1,2}(\vec{r}_1,\vec{r}_2)$$

Y por *La Tercera Ley de Newton* nuevamente:

$$\vec{F}_{1,2} + \vec{F}_{2,1} = m_2\frac{d^2\vec{r}_2(t)}{dt^2} + m_1\frac{d^2\vec{r}_1(t)}{dt^2} = \frac{d^2}{dt^2}(m_1\vec{r}_1 + m_2\vec{r}_2) = 0$$

Como es igual a 0, esto significa que no sufre fuerzas externas, tal y como lo queríamos plantear. Ahora viene la magia del ejercicio, nosotros tenemos una igualdad que es igual a 0, por tanto puedo dividir entre la suma de las masas que se seguira cumpliendo (esto es obvio porque la suma de las masas es distinto de cero), por tanto:

$$\frac{d^2}{dt^2}\left(\frac{m_1\vec{r}_1 + m_2\vec{r}_2}{m_1+m_2}\right) = 0$$

Y curiosamente, si volvemos atrás a la asignatura de F.F.I., recordaremos que esa expresión coincide justo con el centro de masas de un sistema, es decir:

$$\frac{d^2}{dt^2}\vec{R}_{CM} = 0$$

Y como es de esperar, este centro del centro de masas, estará en reposo, por lo que hemos pasado de $\{\vec{r}_1(t),\vec{r}_2(t)\} \rightarrow \{\vec{R}_{CM},\vec{r}(t)\}$, donde $\vec{r}(t) = \vec{r}_2(t) - \vec{r}_1(t)$ que describe la distancia entre los cuerpos (realmente describe la posición del cuerpo 2 respecto al cuerpo 1).

A continuación vamos a realizar una simplificación que será clave en el desarrollo del ejercicio:

$$\vec{F}_{1,2} + \vec{F}_{2,1} = 0 \Longrightarrow \vec{F}_{1,2} = -\vec{F}_{2,1} \Longrightarrow m_2\frac{d^2\vec{r}_2(t)}{dt^2} = -m_1\frac{d^2\vec{r}_2}{dt^2}$$

Como sabemos que $\vec{r}(t) = \vec{r}_2(t) - \vec{r}_1(t)$, entonces podemos expresarlo:

$$\frac{d^2\vec{r}(t)}{dt^2} = \frac{d^2\vec{r}_2(t)}{dt^2} - \frac{d^2\vec{r}_1(t)}{dt^2}$$

$$\frac{d^2\vec{r}(t)}{dt^2} = \frac{\vec{F}_{1,2}}{m_2} - \frac{\vec{F}_{2,1}}{m_1}$$

Y aplicando lo deducido anteriormente podemos generalizarlo a una única fuerza:

$$\frac{d^2}{dt^2}\vec{r}(t) = \vec{F}\left(\frac{1}{m_2} + \frac{1}{m_1}\right)$$

Ahora por ser más cómodo hacemos la siguiente definición:

$$\frac{1}{\mu} = \frac{1}{m_1} + \frac{1}{m_2} \Longrightarrow \mu = \frac{m_1 m_2}{m_1 + m_2}$$

Finalizando así con esta expresión:

$$\mu\frac{d^2\vec{r}(t)}{dt^2} = \vec{F}(\vec{r})$$

Esto se le conoce como *masa reducida*.

$$\mu\frac{d^2\vec{r}(t)}{dt^2} = -G\frac{m_1 m_2}{|\vec{r}(t)|^2}\frac{\vec{r}(t)}{|\vec{r}(t)|}$$

Como $\vec{r}(t) = (x(t), y(t), z(t))$

$$\vec{F}(x(t), y(t), z(t))$$

$$\frac{d^2x(t)}{dt^2} = -\frac{1}{\mu}G\frac{m_1 m_2}{|\vec{r}(t)|^3}x(t) = -G\frac{m_1 m_2}{\mu\left(\sqrt{x^2+y^2+z^2}\right)^3}x(t)$$

$$\frac{d^2y(t)}{dt^2} = -\frac{1}{\mu}G\frac{m_1 m_2}{|\vec{r}(t)|^3}y(t) = -G\frac{m_1 m_2}{\mu\left(\sqrt{x^2+y^2+z^2}\right)^3}y(t)$$

$$\frac{d^2z(t)}{dt^2} = -\frac{1}{\mu}G\frac{m_1 m_2}{|\vec{r}(t)|^3}z(t) = -G\frac{m_1 m_2}{\mu\left(\sqrt{x^2+y^2+z^2}\right)^3}z(t)$$

Pero $z(t) = 0$ ya que suponemos que orbitan por así decir en el mismo ''plano'':

$$\frac{d^2x(t)}{dt^2} = -G\frac{m_1 m_2}{\mu\left(\sqrt{x^2+y^2}\right)^3}x(t) \qquad \frac{d^2y(t)}{dt^2} = -G\frac{m_1 m_2}{\mu\left(\sqrt{x^2+y^2}\right)^3}y(t)$$

Para finalizar, debemos definir la velocidad, que si hacemos memoria:

$$\mu\frac{d^2\vec{r}}{dt^2} = \vec{F} \Longrightarrow \frac{d^2\vec{r}}{dt^2} = \frac{d\vec{v}}{dt} = \frac{\vec{F}}{\mu} = \vec{v}'(t)$$

$$v_x' = -G\frac{m_1 m_2}{\mu\left(\sqrt{x^2+y^2}\right)^3}x(t) \Longrightarrow v_i'(x) = -G\frac{m_1 m_2}{\mu\left(x_i^2+y_i^2\right)^{3/2}}x_i$$

$$v_y' = -G\frac{m_1 m_2}{\mu\left(\sqrt{x^2+y^2}\right)^3}y(t) \Longrightarrow v_i'(y) = -G\frac{m_1 m_2}{\mu\left(x_i^2+y_i^2\right)^{3/2}}y_i$$

$$v'_x(t) = \lim_{\Delta t\to 0} = \frac{v_x(t+\Delta h) - v_x(t)}{\Delta t} \Longrightarrow v'_{x,i} \approx \frac{v_{x,i+1} - v_{x,i}}{\Delta t} \Longrightarrow v_{x,i+1} = v_{x,i} + v'_{x,i}\Delta t$$

$$v'_y(t) = \lim_{\Delta t\to 0} = \frac{v_y(t+\Delta h) - v_y(t)}{\Delta t} \Longrightarrow v'_{y,i} \approx \frac{v_{y,i+1} - v_{y,i}}{\Delta t} \Longrightarrow v_{y,i+1} = v_{y,i} + v'_{y,i}\Delta t$$

Y de la misma manera con la posición:

$$x'(t) = \lim_{\Delta t\to 0} = \frac{x(t+\Delta h) - x(t)}{\Delta t} \Longrightarrow x'_i \approx \frac{x_{i+1} - x_i}{\Delta t} \Longrightarrow x_{i+1} = x_i + v_{x,i}\Delta t$$

$$y'(t) = \lim_{\Delta t\to 0} = \frac{y(t+\Delta h) - y(t)}{\Delta t} \Longrightarrow y'_i \approx \frac{y_{i+1} - y_i}{\Delta t} \Longrightarrow y_{i+1} = y_i + v_{y,i}\Delta t$$

```
import numpy as np
import matplotlib.pyplot as plt

#Variables temporales
tf=31536000*1.5           #Tiempo final de 1'5 años
t0=0                      #Tiempo inicial
dt = (24*3600)            #Delta t de un día
N= int((tf-t0)/(dt))      #Número de intervalos

#Variables espaciales
x0 = 1.521e11             #Posición en el afelio [m] (eje horizontal)
y0 = 0                    #Posición en el afelio [m] (eje vertical)
vx0 = 0                   #v0 [m/s](eje horizontal)
vy0 = 2.929e4             #v0 [m/s] (en el eje vertical, pues es tangente a la␣
 ↪trayectoria en el afelio)

#Variables físicas
G =  6.67e-11             #Constante de gravitación universal
M = 1.989e30              #Masa del Sol
m = 5.972168e24           #Masa de la Tierra
mu = (M*m)/(M+m)          #Masa reducida de la Tierra  y el Sol
```

2.1. Algoritmo de Euler:

2.1.1. Creamos los arrays:

```
t=np.linspace(t0,tf,N+1)
xeuler=np.zeros(N+1)
yeuler=np.zeros(N+1)
vxeuler=np.zeros(N+1)
vyeuler=np.zeros(N+1)
r=np.zeros(N+1)
```

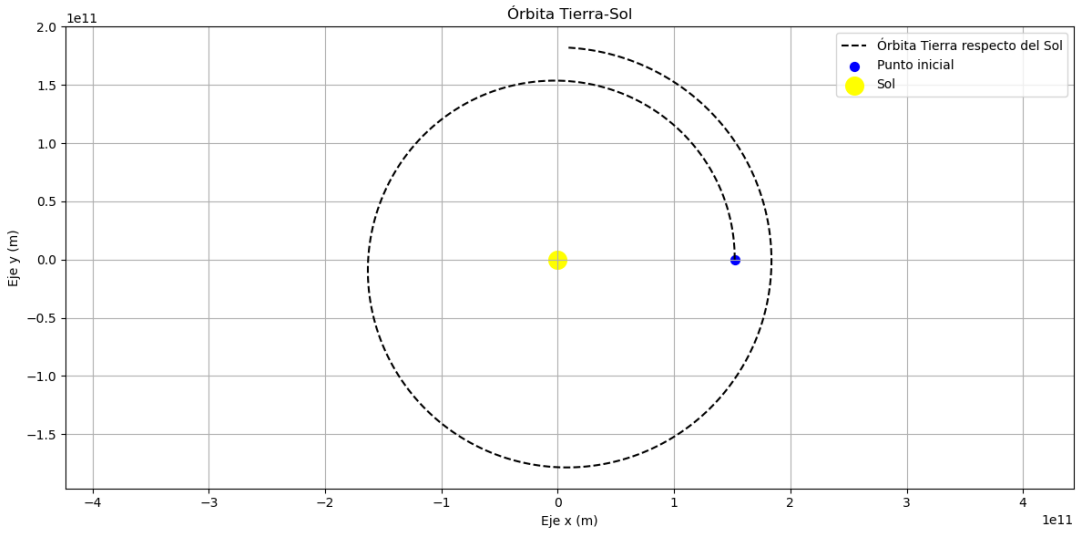
2.1.2. Funciones:

```
xeuler[0]=x0
yeuler[0]=y0
vxeuler[0]=vx0
vyeuler[0]=vy0
```

```
for i in range (0,N):
    r[i]=xeuler[i]**2+yeuler[i]**2
    xeuler[i+1]=xeuler[i]+vxeuler[i]*dt
    yeuler[i+1]=yeuler[i]+vyeuler[i]*dt
    vxeuler[i+1]=vxeuler[i]-G*((M*m)/(mu*(r[i])**(3/2)))*xeuler[i]*dt
    vyeuler[i+1]=vyeuler[i]-G*((M*m)/(mu*(r[i])**(3/2)))*yeuler[i]*dt
```

2.1.3. Gráficas:

```
plt.figure(figsize=(12,6))
plt.plot(xeuler,yeuler,label='Órbita Tierra respecto del␣
 ↪Sol',color='black',linestyle='dashed')
plt.scatter(x0,0,label='Punto inicial',color='blue',marker='o',s=50)
plt.scatter(0,0,label='Sol',color='yellow',marker='o',s=200)
plt.title("Órbita Tierra-Sol")
plt.xlabel("Eje x (m)")
plt.ylabel("Eje y (m)")
plt.grid(True)
plt.axis("equal")
plt.tight_layout();plt.legend();plt.show()
```



Como podemos observar, si hacemos que la Tierra haga su traslación durante 1'5 años el error es más que evidente, dándonos una falsa idea de que la Tierra escapa del pozo gravitacional del Sol y se aleja lentamente hasta acabar con nuestra existencia, por suerte para nosotros estábamos trabajando con el *Algoritmo de Euler*, por lo que será mejor si trabajamos con el modificado.

2.2. Algoritmo de Euler modificado:

Ahora, vamos a usar el algoritmo de Euler modificado, que su estructura es muy similar al normal:

$$x_{i+1} = x_i + v_{x,i+1}\Delta t$$
$$y_{i+1} = y_i + v_{y,i+1}\Delta t$$

$$v_{x,i+1} = v_{x,i} + v'_{x,i}\Delta t$$
$$v_{y,i+1} = v_{y,i} + v'_{y,i}\Delta t$$

2.2.1. Creamos los arrays:

```
t=np.linspace(t0,tf,N+1)
xeulermod=np.zeros(N+1)
yeulermod=np.zeros(N+1)
vxeulermod=np.zeros(N+1)
vyeulermod=np.zeros(N+1)
r=np.zeros(N+1)
```

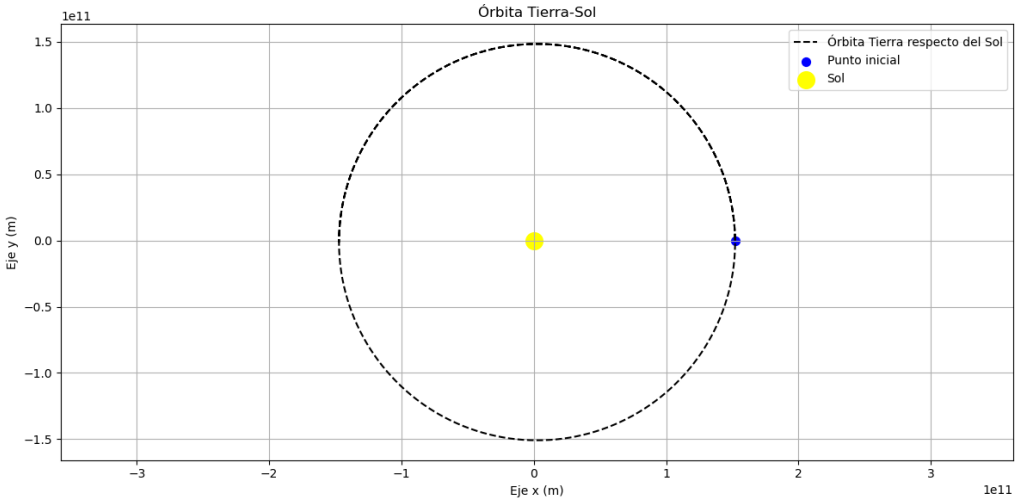
2.2.2. Funciones:

```
xeulermod[0]=x0
yeulermod[0]=y0
vxeulermod[0]=vx0
vyeulermod[0]=vy0
```

```
for i in range (0,N):
    r[i]=xeulermod[i]**2+yeulermod[i]**2
    vxeulermod[i+1]=vxeulermod[i]-G*((M*m)/(mu*(r[i])**(3/2)))*xeulermod[i]*dt
    vyeulermod[i+1]=vyeulermod[i]-G*((M*m)/(mu*(r[i])**(3/2)))*yeulermod[i]*dt
    xeulermod[i+1]=xeulermod[i]+vxeulermod[i+1]*dt
    yeulermod[i+1]=yeulermod[i]+vyeulermod[i+1]*dt
```

2.2.3. Gráficas:

```
plt.figure(figsize=(12,6))
plt.plot(xeulermod,yeulermod,label='Órbita Tierra respecto del␣
 ↪Sol',color='black',linestyle='dashed')
plt.scatter(x0,0,label='Punto inicial',color='blue',marker='o',s=50)
plt.scatter(0,0,label='Sol',color='yellow',marker='o',s=200)
plt.title("Órbita Tierra-Sol")
plt.xlabel("Eje x (m)")
plt.ylabel("Eje y (m)")
plt.grid(True)
plt.axis("equal")
plt.tight_layout();plt.legend();plt.show()
```



2.3. Periodo:

```
epsx=2e9
epsy=2e9 #Podéis usar 1 en vez de N//2 pero Chicón usó N//2, da lo mismo.
for i in range(N//2, N+1):
    if abs(xeulermod[i] - xeulermod[0]) < epsx and abs(yeulermod[i] - yeulermod[0])␣
↪< epsy:
        T = t[i] - t[0]
        break

T_periodo = T / (86400)
print(f"El periodo de órbita es: {T_periodo:.2f} días.")
```

```
El periodo de órbita es: 365.33 días.
```

2.4. Conservación del momento angular:

Para calcular la conservación del momento angular:

$$\vec{\tau} = \vec{r} \times \vec{F} = \vec{r} \times \mu \frac{d^2\vec{r}}{dt^2} = \frac{d}{dt}\left(\mu\vec{r} \times \frac{d\vec{r}}{dt}\right) = \frac{d}{dt}\vec{L} = 0$$

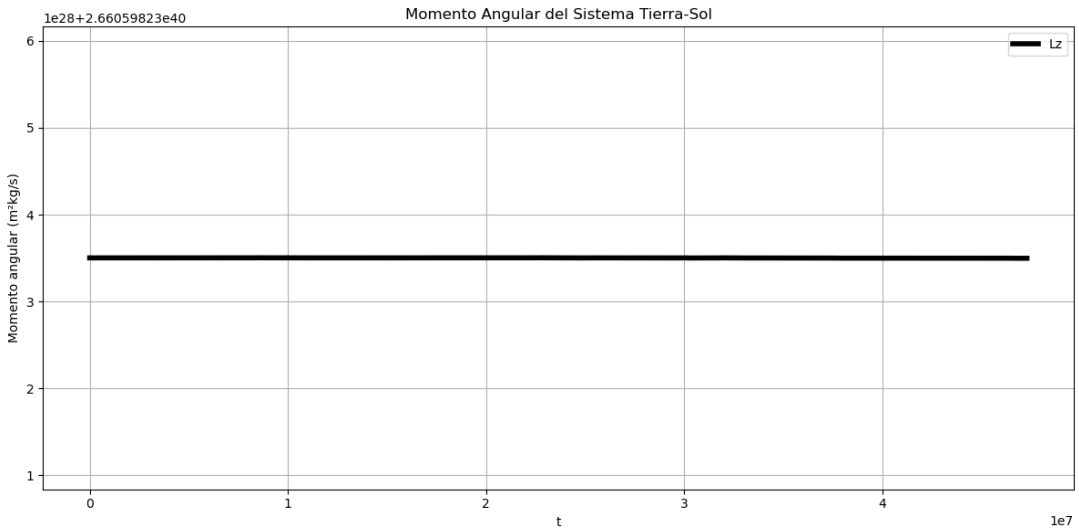
Por tanto queda demostrado que el momento angular es constante:

$$\vec{L} = \mu\vec{r} \times \frac{d\vec{r}}{dt} = \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ \mu x & \mu y & 0 \\ v_x & v_y & 0 \end{vmatrix} = (0, 0, \mu x \cdot v_y - \mu y \cdot v_x)$$

```
Lz=np.zeros(N+1)
```

```
for i in range(0,N+1):
    Lz[i]=mu*xeulermod[i]*vyeulermod[i]-mu*yeulermod[i]*vxeulermod[i]
```

```
plt.figure(figsize=(12,6))
plt.plot(t, Lz, label = "Lz",color='black',lw=4)
plt.title('Momento Angular del Sistema Tierra-Sol')
plt.xlabel('t')
plt.ylabel('Momento angular (m²kg/s)')
plt.grid();plt.tight_layout(); plt.legend(); plt.show();
lz=mu*((xeulermod[0]*vyeulermod[0]) - (yeulermod[0]*vxeulermod[0]))
print (f"El valor en el 0 del momento angular es de {lz:.2g} m²kg/s")
```



El valor en el 0 del momento angular es de 2.7e+40 m^2kg/s

2.5. Conservación de la energía:

Ahora podemos calcular la energía cinética y potencial en una órbita como es el caso:

- **Energía Cinética:**

$$K = \frac{1}{2} m_T v_T^2$$

donde la velocidad de una órbita sabemos calcularla:

$$F = m_T a_c \Longrightarrow G\frac{M_S m_T}{r^2} = m_T \frac{v^2}{r} \Longrightarrow v^2 = G\frac{M_S}{r}$$

$$K = \frac{1}{2} m_T v_T^2 = \frac{1}{2} m_T (v_x^2 + v_y^2) \qquad K = \frac{1}{2} m_T \left(G\frac{M_S}{r} \right)$$

- **Energía Potencial:**

$$U = -\int -G\frac{M_S m_T}{r^2} dr$$

$$U = -G\frac{M_S m_T}{\sqrt{x^2+y^2}} \qquad U = -G\frac{M_S m_T}{r}$$

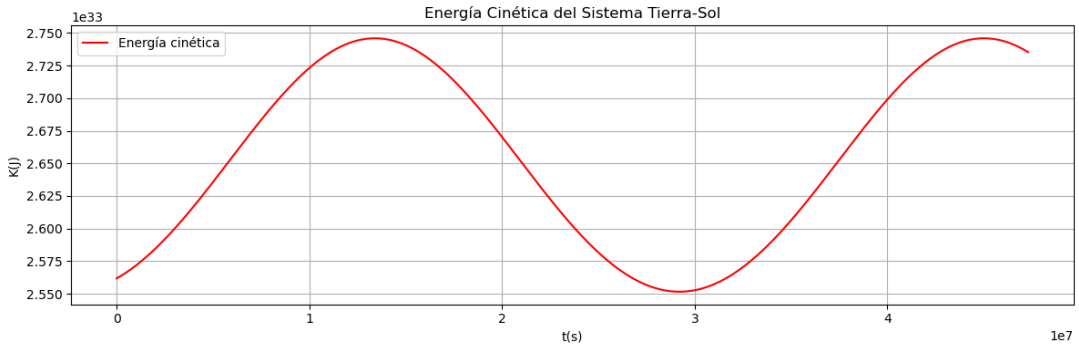
2.5.1. Creamos los arrays:

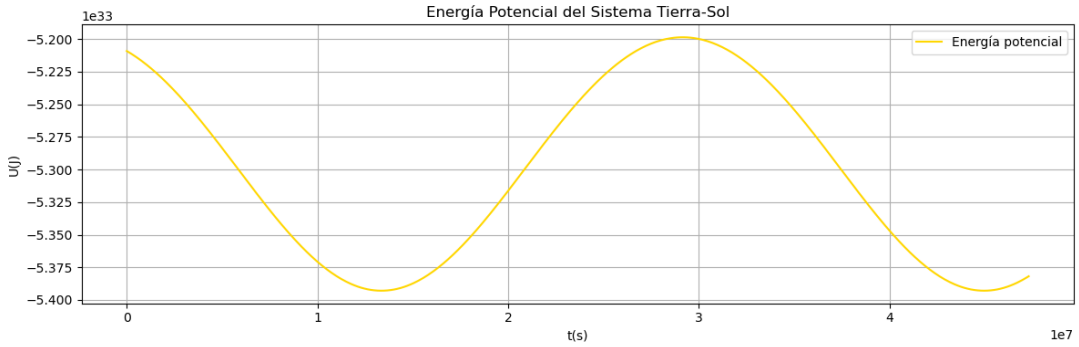
```
Ekeulermod=np.zeros(N+1)
Eueulermod=np.zeros(N+1)
Emeulermod=np.zeros(N+1)
```

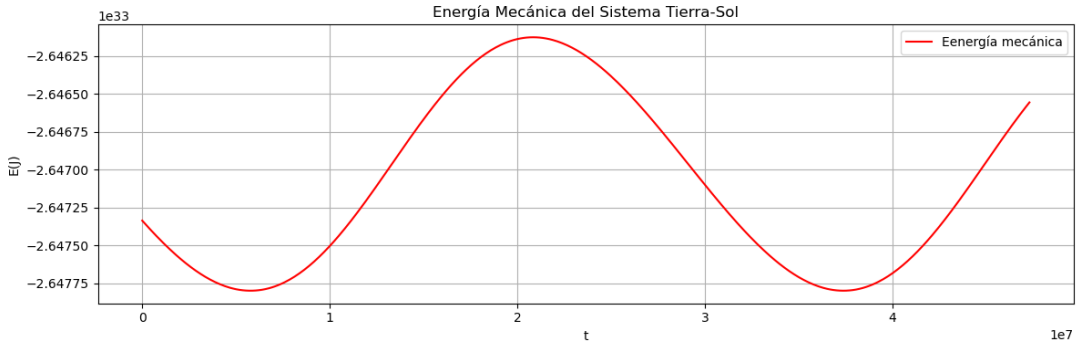
2.5.2. Funciones:

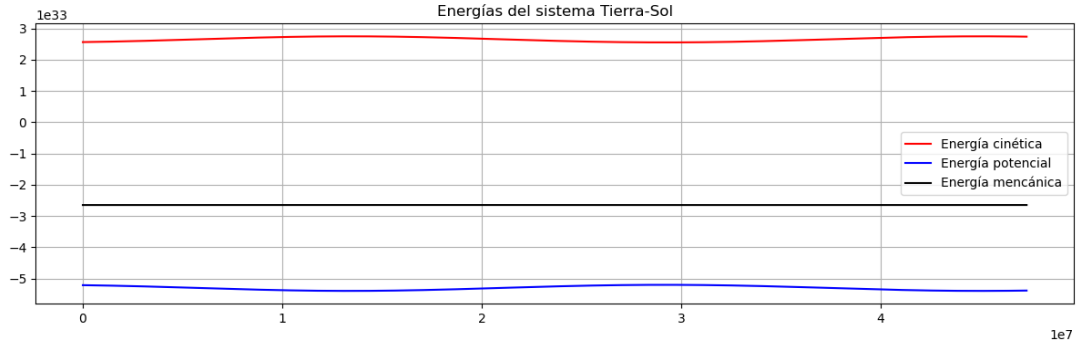
```
for i in range(0,N+1):
    Ekeulermod[i]=(1/2)*m*(vxeulermod[i]**2 + vyeulermod[i]**2)
    Eueulermod[i]=-G*(M*m)/(np.sqrt(xeulermod[i]**2+yeulermod[i]**2))
    Emeulermod[i]=Ekeulermod[i]+Eueulermod[i]
```

```
plt.figure(figsize=(12,4))
plt.plot(t, Ekeulermod, label = "Energía cinética",color='red'); plt.title('Energía␣
 ↪Cinética del Sistema Tierra-Sol')
plt.xlabel('t(s)'); plt.ylabel('K(J)');plt.grid();plt.tight_layout(); plt.legend();␣
 ↪plt.show();

plt.figure(figsize=(12,4))
plt.plot(t, Eueulermod, label = "Energía potencial",color='gold'); plt.
 ↪title('Energía Potencial del Sistema Tierra-Sol')
plt.xlabel('t(s)'); plt.ylabel('U(J)');plt.grid();plt.tight_layout(); plt.legend();␣
 ↪plt.show();

plt.figure(figsize=(12,4))
plt.plot(t, Emeulermod, label = "Eenergía mecánica",color='red')
plt.title('Energía Mecánica del Sistema Tierra-Sol')
plt.xlabel('t');plt.ylabel('E(J)');plt.grid();plt.tight_layout(); plt.legend(); plt.
 ↪show();

plt.figure(figsize=(12,4))
plt.plot(t, Ekeulermod, label = "Energía cinética",color='red')
plt.plot(t, Eueulermod, label = "Energía potencial",color='blue')
plt.plot(t, Emeulermod, label = "Energía mencánica",color='black')
plt.title('Energías del sistema Tierra-Sol')
plt.grid();plt.tight_layout(); plt.legend(); plt.show();
```









Y como se muestra en la imagen, la energía se conserva satisfactoriamente.

2.6. Coordenadas en el perihelio, longitud semiejes y excentricidad:

```
# Coordenadas del perihelio
indice_xmax = np.argmax(xeulermod) # se halla el índice que maximiza x
#(recuerde que el afelio, cuyo |x| es el mayor valor, es negativo) para hallar la␣
 ↪posición del perihelio
indice_ymax = np.argmax(yeulermod) # se halla el índice que maximiza y para hallar␣
 ↪la velocidad en el perihelio (máxima y en el eje y)
perihelio = [xeulermod[indice_xmax], 0]
print("Las coordenadas del perihelio son:", perihelio)
print("La velocidad en el perihelio es:", [round(vxeulermod[indice_ymax], 3),␣
 ↪round(vyeulermod[indice_ymax], 3)], "m/s")
print()
# Longitud de los semiejes
semieje_mayor = (xeulermod[np.argmax(xeulermod)]-xeulermod[np.argmin(xeulermod)])/2
semieje_menor = (yeulermod[np.argmax(yeulermod)]-yeulermod[np.argmin(yeulermod)])/2
print("El semieje mayor mide", round(semieje_mayor, 2), "m")
print("El semieje menor mide", round(semieje_menor, 2), "m")
print()
# Excentricidad
e = np.sqrt(1-(semieje_menor/semieje_mayor)**2)
print("La excentricidad de la órbita terrestre es:", round(e, 5))
```

```
Las coordenadas del perihelio son: [152104701180.4843, 0]
La velocidad en el perihelio es: [-30017.806, 216.625] m/s

El semieje mayor mide 149657424713.63 m
El semieje menor mide 149641733951.27 m

La excentricidad de la órbita terrestre es: 0.01448
```

3. Extra: Péndulo

Tenemos que un péndulo sigue la siguiente expresión:

$$\vec{F} = m\vec{a} = -mgl\sin\theta \Longrightarrow \frac{d^2s}{dt^2} = l\frac{d^2\theta}{dt^2} \Longrightarrow \frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin\theta$$

Sabmos que $w = \frac{d\theta}{dt}$ y que $\alpha = \frac{dw}{dt} = \frac{d^2\theta}{dt^2}$, por lo que:

$$\theta'(t) = w(t) = \lim_{\Delta t\to 0}\frac{\theta(t+\Delta h)-\theta(t)}{\Delta t} \Longrightarrow w_i \approx \frac{\theta_{i+1}-\theta_i}{\Delta t} \Longrightarrow \theta_{i+1} = \theta_i + w_i\Delta t$$

$$\theta''(t) = w'(t) = \lim_{\Delta t\to 0}\frac{w(t+\Delta h)-w(t)}{\Delta t} \Longrightarrow w_i' \approx \frac{w_{i+1}-w_i}{\Delta t} \Longrightarrow w_{i+1} = w_i + w_i'\Delta t$$

Donde usando el algoritmo de euler modificado usamos las siguientes expresiones:

$$w_{i+1} = w_i + w_i'\Delta t \Longrightarrow w_{i+1} = w_i - \frac{g}{l}\sin\theta_i\Delta t$$

$$\theta_{i+1} = \theta_i + w_{i+1}\Delta t$$

```
import numpy as np
import matplotlib.pyplot as plt
```

```
t0=0
tf=10
N=int(1000)
dt=(tf-t0)/(N)
m=0.5
g=9.81
l=1
```

Creamos los arrays:

```
t=np.linspace(t0,tf,N+1)
theta_eulermod=np.zeros(N+1)
w_eulermod=np.zeros(N+1)
```

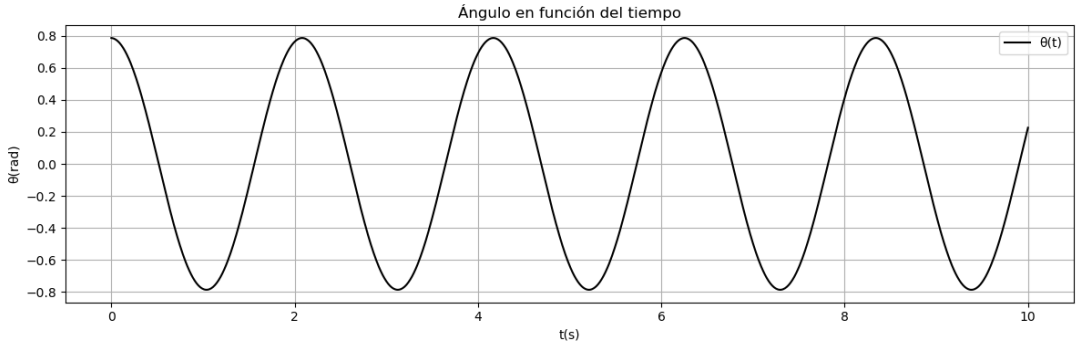
Funciones:

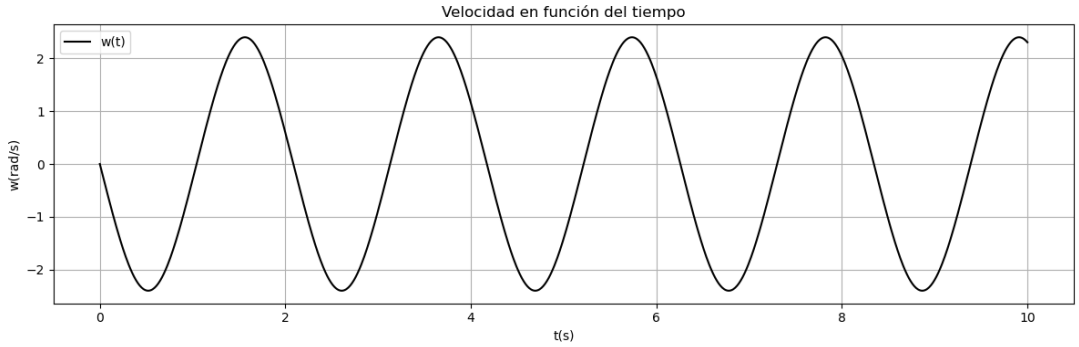
```
w_eulermod[0]=0
theta_eulermod[0]=np.radians(45)
```

```
for i in range(0,N):
    w_eulermod[i+1]=w_eulermod[i]-(g/l)*np.sin(theta_eulermod[i])*dt
    theta_eulermod[i+1]=theta_eulermod[i]+w_eulermod[i+1]*dt
```

Gráficas:

```
plt.figure(figsize=(12,4))
plt.plot(t,theta_eulermod,label='θ(t)',color='black')
plt.title("Ángulo en función del tiempo")
plt.xlabel("t(s)")
plt.ylabel("θ(rad)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()

plt.figure(figsize=(12,4))
plt.plot(t,w_eulermod,label='w(t)',color='black')
plt.title("Velocidad en función del tiempo")
plt.xlabel("t(s)")
plt.ylabel("w(rad/s)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```





Podemos ir más allá extrapolándolo a las coordenadas cartesianas:

$$y = L\cos\theta$$

$$x = L\sin\theta$$

$$v = Lw$$

Pero si queremos saber la altura real en función del tiempo no tenemos más que:

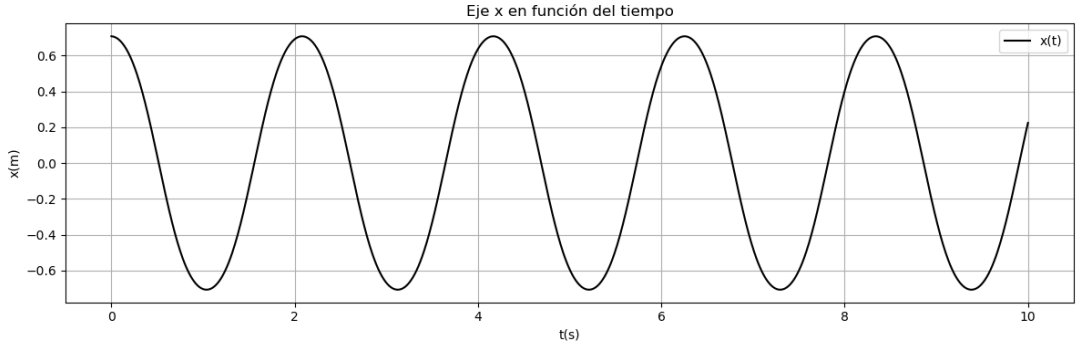
$$h = (1 - cos\theta)l \Longrightarrow h = L - y(\theta)$$

Que como podemos esperar, sus posiciones en los ejes oscilarán en función del tiempo:

```
x_eulermod=np.zeros(N+1)
y_eulermod=np.zeros(N+1)
v_eulermod=np.zeros(N+1)
```

```
for i in range(0,N+1):
    x_eulermod[i]=l*np.sin(theta_eulermod[i])
    y_eulermod[i]=l*np.cos(theta_eulermod[i])
    v_eulermod[i]=l*w_eulermod[i]
```

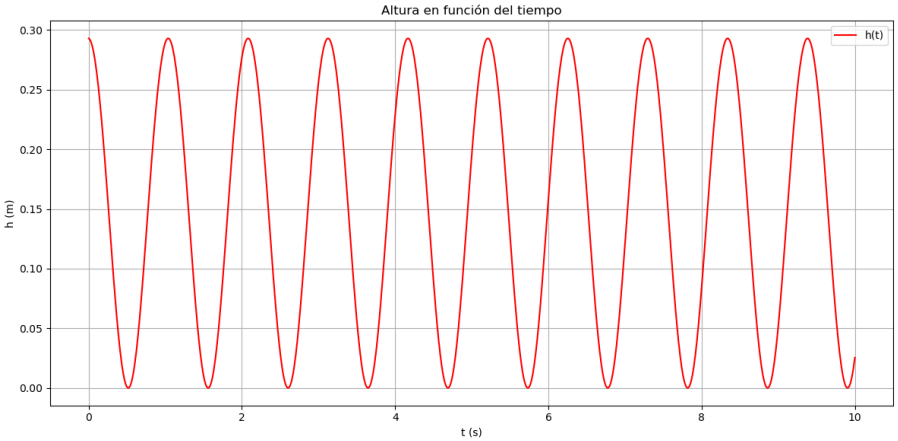
```
plt.figure(figsize=(12,4))
plt.plot(t,x_eulermod,label='x(t)',color='black')
plt.title("Eje x en función del tiempo")
plt.xlabel("t(s)")
plt.ylabel("x(m)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()


plt.figure(figsize=(12,4))
plt.plot(t,y_eulermod,label='y(t)',color='red')
plt.title("Eje y en función del tiempo")
plt.xlabel("t(s)")
plt.ylabel("y(m)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```



Observamos que, cuando la y es máxima, indica que está en el punto más bajo, totalmente vertical, y en el más bajo cuando está en el punto más alto, esto lo podemos ver mejor si le restamos el largo de la cuerda y vemos comparamos desde el punto (0,0) por así decir.

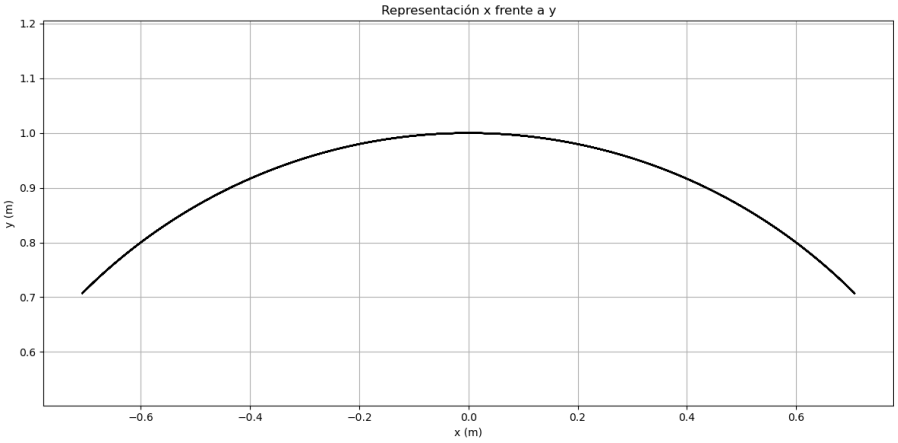
```
h_eulermod=l-y_eulermod
```

```
plt.figure(figsize=(12,6))
plt.plot(t,h_eulermod,label='h(t)',color='red')
plt.title("Altura en función del tiempo")
plt.xlabel("t (s)")
plt.ylabel("h (m)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```



Vamos a ponernos creativos y vamos a representar la gráfica en función de x e y:

```
plt.figure(figsize=(12,6))
plt.plot(x_eulermod,y_eulermod,color='black')
plt.title("Representación x frente a y")
plt.xlabel("x (m)")
plt.ylabel("y (m)")
plt.grid(True)
plt.axis('equal');plt.tight_layout();plt.show()
```



Ahora vamos a comprobar su periodo:

```
epsx=0.0001
epsy=0.0001
for i in range(1, N+1):
    if abs(x_eulermod[i] - x_eulermod[0]) < epsx and abs(y_eulermod[i] -␣
 ↪y_eulermod[0]) < epsy:
        T = t[i] - t[0]
        break
T_teorico=2*np.pi*np.sqrt(l/g)
print(f"El periodo teórico es: {T_teorico:.2f} s")
print(f"El periodo experimental es: {T:.2f} s")
```

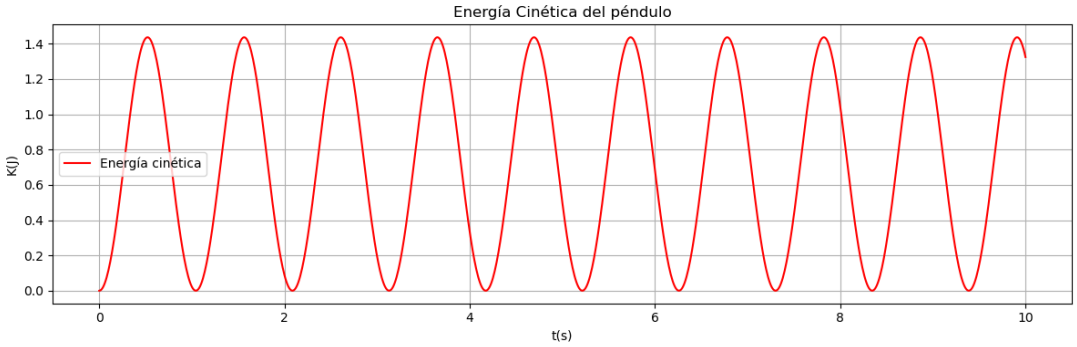
```
El periodo teórico es: 2.01 s
El periodo experimental es: 2.08 s
```

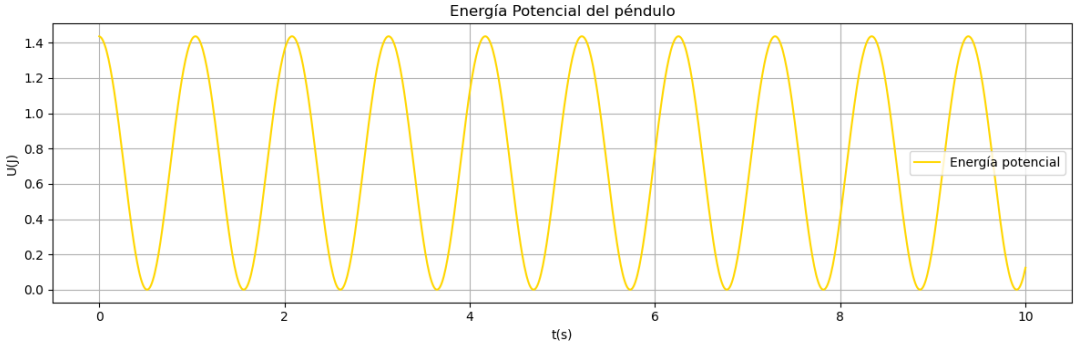
Y por último vamos a calcular la energía:

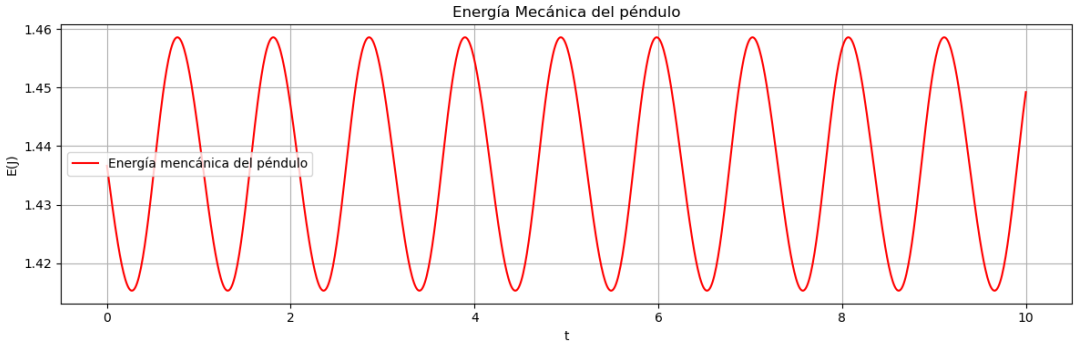
```
Keulermod=np.zeros(N+1)
Ueulermod=np.zeros(N+1)
Emeulermod=np.zeros(N+1)
```

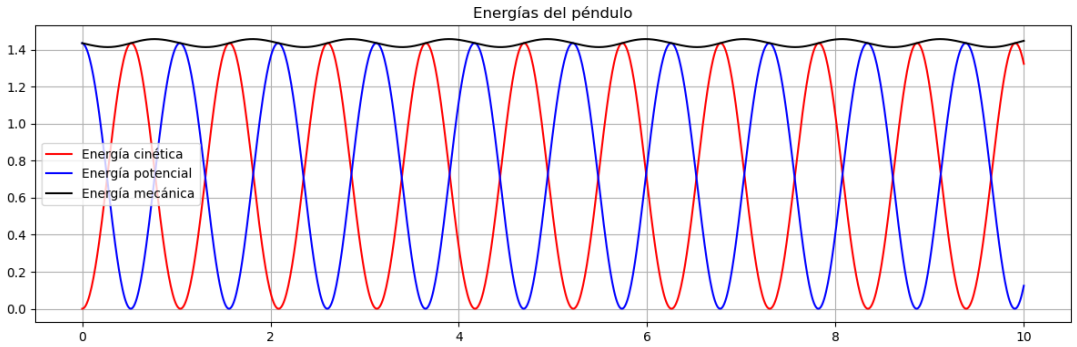
```
for i in range (0,N+1):
    Keulermod[i]=(1/2)*m*v_eulermod[i]**2
    Ueulermod[i]=m*g*h_eulermod[i]
    Emeulermod[i]=Keulermod[i]+Ueulermod[i]
```

```
plt.figure(figsize=(12,4))
plt.plot(t, Keulermod, label = "Energía cinética",color='red'); plt.title('Energía␣
 ↪Cinética del péndulo')
plt.xlabel('t(s)'); plt.ylabel('K(J)');plt.grid();plt.tight_layout(); plt.legend();␣
 ↪plt.show();

plt.figure(figsize=(12,4))
plt.plot(t, Ueulermod, label = "Energía potencial",color='gold'); plt.
 ↪title('Energía Potencial del péndulo')
plt.xlabel('t(s)'); plt.ylabel('U(J)');plt.grid();plt.tight_layout(); plt.legend();␣
 ↪plt.show();

plt.figure(figsize=(12,4))
plt.plot(t, Emeulermod, label = "Energía mencánica del péndulo",color='red')
plt.title('Energía Mecánica del péndulo')
plt.xlabel('t');plt.ylabel('E(J)');plt.grid();plt.tight_layout(); plt.legend(); plt.
 ↪show();

plt.figure(figsize=(12,4))
plt.plot(t, Keulermod, label = "Energía cinética",color='red')
plt.plot(t, Ueulermod, label = "Energía potencial",color='blue')
plt.plot(t, Emeulermod, label = "Energía mecánica",color='black')
plt.title('Energías del péndulo')
plt.grid();plt.tight_layout(); plt.legend(); plt.show();
```


Energía Cinética del péndulo
Energía cinética
K(J)
t(s)


Energía Potencial del péndulo
Energía potencial
U(J)
t(s)


Energía Mecánica del péndulo
Energía mencánica del péndulo
E(J)
t


Energías del péndulo
Energía cinética
Energía potencial
Energía mecánica

4. Extra: Muelle en dos dimensiones

Una partícula de masa m se mueve en el plano XY bajo la acción de una fuerza $\vec{F}$ dada por:

$$\vec{F} = -k_x x\hat{\imath} - k_y y\hat{\jmath}$$

en donde (x, y) denota la posición de la partícula e $\hat{\imath}$ y $\hat{\jmath}$ son los vectores unitarios sobre los ejes X, Y, respectivamente. En $t = 0$ se encuentra en (x_0, y_0), y su velocidad es $(v_{x,0}, v_{y,0})$.

Calcule, usando el **Método de Euler Modificado**, el movimiento de la partícula para $t > 0$.

1) **Haga en papel, un resumen de las ecuaciones del problema que va a resolver numéricamente y escriba claramente las líneas principales del algoritmo con las que lo va a resolver. Indiquetambién todas las cuestiones que son necesarias tener en cuenta para obetener un buen resultado en el cálculo.**

Primero de todo, vamos a empezar a descomponer el problema en las coordenadas correspondientes para operar de manera más sencilla:

$$\begin{aligned} F_x &= -k_x x & F_y &= -k_y y \\ ma_x &= -k_x x & ma_y &= -k_y y \\ m\frac{d^2x}{dt^2} &= -k_x x & m\frac{d^2y}{dt^2} &= -k_y y \\ \frac{d^2x}{dt^2} = x''(t) = v_x'(t) &= -\frac{k_x}{m}x(t) & \frac{d^2y}{dt^2} = y''(t) = v_y'(t) &= -\frac{k_y}{m}y(t) \end{aligned}$$

Ahora por otro lado resolvemos los algoritmos a utilizar:

$$x'(t) = \lim_{\Delta t\to 0}\frac{x(t+\Delta t)-x(t)}{\Delta t} \Longrightarrow x_i' \approx \frac{x_{i+1}-x_i}{\Delta t} \Longrightarrow x_{i+1} = x_i + x_i'\Delta t \Longrightarrow x_{i+1} = x_i + v_{x,i}\Delta t$$

$$y'(t) = \lim_{\Delta t\to 0}\frac{y(t+\Delta t)-y(t)}{\Delta t} \Longrightarrow y_i' \approx \frac{y_{i+1}-y_i}{\Delta t} \Longrightarrow y_{i+1} = y_i + y_i'\Delta t \Longrightarrow y_{i+1} = y_i + v_{y,i}\Delta t$$

Lo mismo con la velocidad:

$$v_x'(t) = \lim_{\Delta t\to 0}\frac{v_x(t+\Delta t)-v_x(t)}{\Delta t} \Longrightarrow v_{x,i}' \approx \frac{v_{x,i+1}-v_{x,i}}{\Delta t} \Longrightarrow v_{x,i+1} = v_{x,i}+v_{x,i}'\Delta t \Longrightarrow v_{x,i+1} = v_{x,i}+v_{x,i}'\Delta t$$

$$v_y'(t) = \lim_{\Delta t\to 0}\frac{v_y(t+\Delta t)-v_y(t)}{\Delta t} \Longrightarrow v_{y,i}' \approx \frac{v_{y,i+1}-v_{y,i}}{\Delta t} \Longrightarrow v_{y,i+1} = v_{y,i}+v_{y,i}'\Delta t \Longrightarrow v_{y,i+1} = v_{y,i}+v_{y,i}'\Delta t$$

Pero el algoritmo de Euler Modificado se caracteriza por seguir las siguientes ecuaciones:

$$\begin{aligned} x_{i+1} &= x_i + v_{x,i+1}\Delta t & y_{i+1} &= y_i + v_{y,i+1}\Delta t \\ v_{x,i+1} &= v_{x,i} + v_{x,i}'\Delta t & v_{y,i+1} &= v_{y,i} + v_{y,i}'\Delta t \end{aligned}$$

De manera que, si las juntamos con las expresiones demostradas anteriormente:

$$\begin{aligned} x_{i+1} &= x_i + v_{x,i+1}\Delta t & y_{i+1} &= y_i + v_{y,i+1}\Delta t \\ v_{x,i+1} &= v_{x,i} - \frac{k_x}{m}x_i\Delta t & v_{y,i+1} &= v_{y,i} - \frac{k_x}{m}y_i\Delta t \end{aligned}$$

2) **Calcule y represente gráficamente la trayectoria en el plano XY.**

```
import numpy as np
import matplotlib.pyplot as plt
```

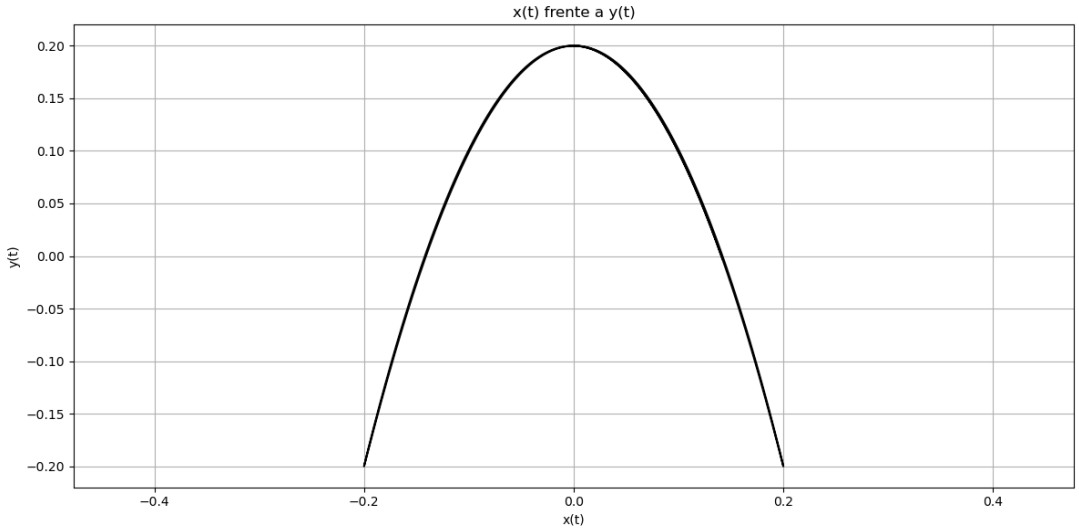
```
m=0.04
kx=2e-4
ky=8e-4
x0=0
y0=0.2
vx0=np.sqrt(2)/100
vy0=0
t0=0
tf=100
N=int(1000)
dt=(tf-t0)/(N)
```

```
t=np.linspace(t0,tf,N+1)
x_eulermod=np.zeros(N+1)
y_eulermod=np.zeros(N+1)
vx_eulermod=np.zeros(N+1)
vy_eulermod=np.zeros(N+1)
```

```
x_eulermod[0]=x0
y_eulermod[0]=y0
vx_eulermod[0]=vx0
vy_eulermod[0]=vy0
```

```
for i in range (0,N):
    vx_eulermod[i+1]=vx_eulermod[i]-(kx/m)*x_eulermod[i]*dt
    vy_eulermod[i+1]=vy_eulermod[i]-(ky/m)*y_eulermod[i]*dt
    x_eulermod[i+1]=x_eulermod[i]+vx_eulermod[i+1]*dt
    y_eulermod[i+1]=y_eulermod[i]+vy_eulermod[i+1]*dt
```

```
plt.figure(figsize=(12,6))
plt.plot(x_eulermod,y_eulermod,color='black')
plt.title("x(t) frente a y(t)")
plt.xlabel("x(t)")
plt.ylabel("y(t)")
plt.grid(True)
plt.axis('equal');plt.tight_layout();plt.show()
```


x(t) frente a y(t)
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
y(t)
−0.4
−0.2
0.0
0.2
0.4
x(t)

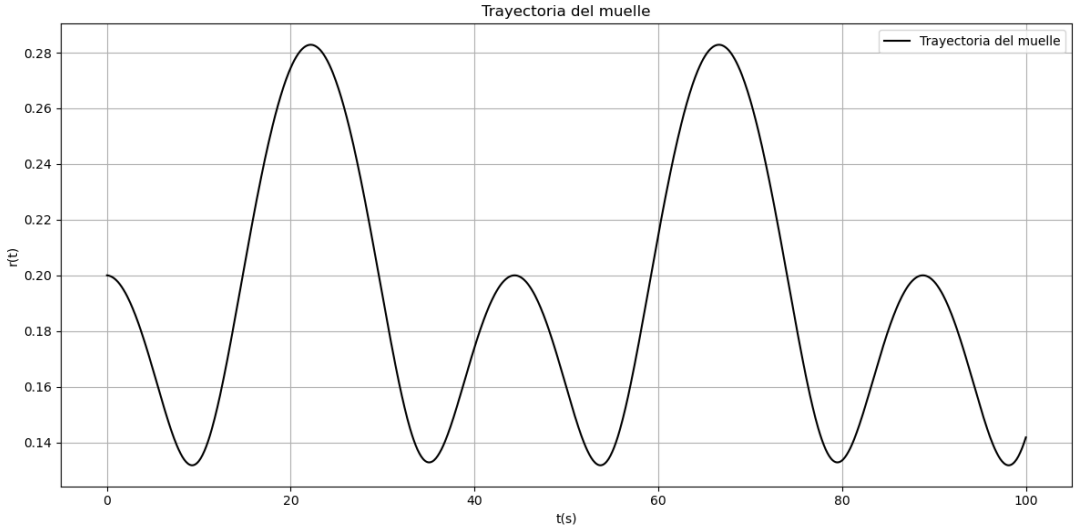
3) **Comente todas las características del movimiento que le parezcan relevantes**

Podemos complementarlo con una gráfica que describe la distancia al origen del muelle:

```
r=np.zeros(N+1)
```

```
for i in range (0,N+1):
    r[i]=np.sqrt(x_eulermod[i]**2+y_eulermod[i]**2)
```

```
plt.figure(figsize=(12,6))
plt.plot(t,r,label='Trayectoria del muelle',color='black')
plt.title("Trayectoria del muelle")
plt.xlabel("t(s)")
plt.ylabel("r(t)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```



A parte de esta, a destacar podemos graficar la energía cinética, potencial y mecánica para comprobar que se conserva la energía, siendo:

$$K = \frac{1}{2}m\left(v_x^2 + v_y^2\right)$$

$$U = -\int -k_x x dx - \int -k_y y dy = \frac{1}{2}k_x x^2 + \frac{1}{2}k_y y^2$$

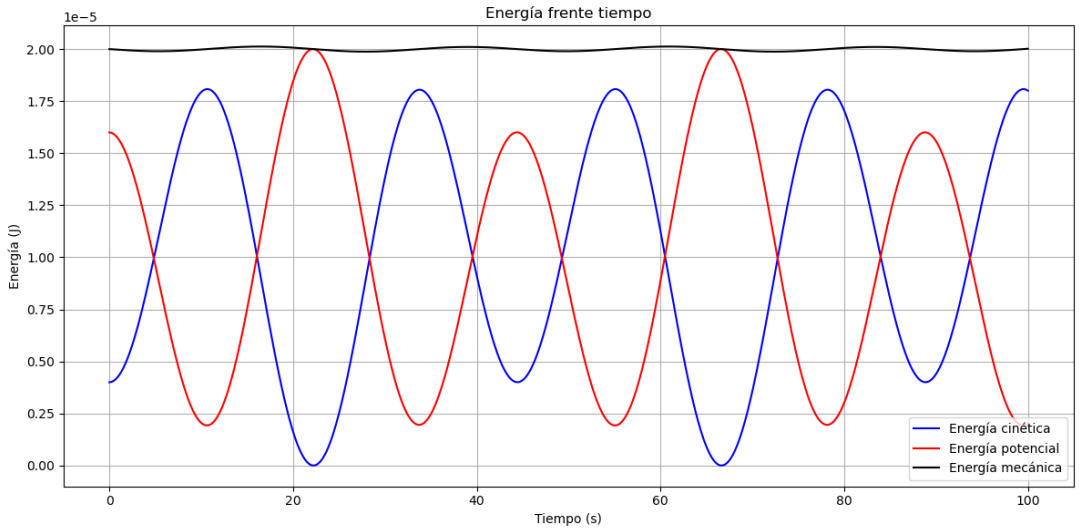
Consideramos la constante C como 0 por simplicidad.

$$E_m = K + U = \frac{1}{2}m\left(v_x^2 + v_y^2\right) + \frac{1}{2}\left(k_x x^2 + k_y y^2\right)$$

```
K=np.zeros(N+1)
U=np.zeros(N+1)
Em=np.zeros(N+1)
```

```
for i in range (0,N+1):
    K[i]=(1/2)*m*(vx_eulermod[i]**2+vy_eulermod[i]**2)
    U[i]=(1/2)*(kx*x_eulermod[i]**2+ky*y_eulermod[i]**2)
    Em[i]=K[i]+U[i]
```

```
plt.figure(figsize=(12,6))
plt.plot(t,K,label='Energía cinética',color='blue')
plt.plot(t,U,label='Energía potencial',color='red')
plt.plot(t,Em,label='Energía mecánica',color='black')
plt.title("Energía frente tiempo")
plt.xlabel("Tiempo (s)")
plt.ylabel("Energía (J)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```



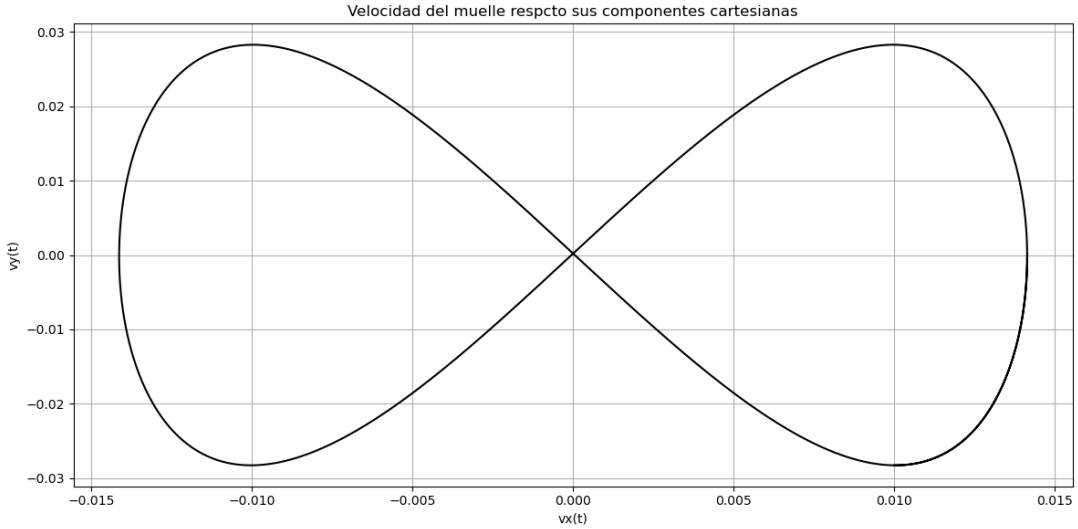
Por otro lado tenemos el periodo:

[88]:
```
epsx=0.001
epsy=0.001
for i in range (1,N+1):
    if abs(x_eulermod[i]-x_eulermod[0])<epsx and␣
 ↪abs(y_eulermod[i]-y_eulermod[0])<epsy:
        T=t[i]
        break
print (f" T aproximado = {T:.3f} seg")
```

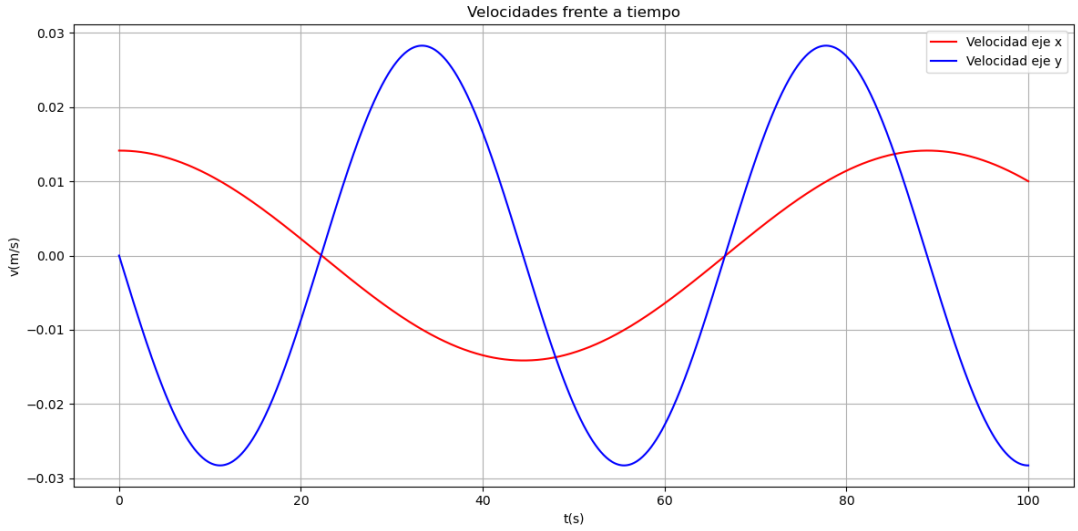
```
 T aproximado = 44.400 seg
```

4) **¿Hay algún momento donde la velocidad sea 0?**

```
plt.figure(figsize=(12,6))
plt.plot(vx_eulermod,vy_eulermod,color='black')
plt.title("Velocidad del muelle respcto sus componentes cartesianas")
plt.xlabel("vx(t)")
plt.ylabel("vy(t)")
plt.grid(True)
plt.tight_layout();plt.show()
```




```
plt.figure(figsize=(12,6))
plt.plot(t,vx_eulermod,label='Velocidad eje x',color='red')
plt.plot(t,vy_eulermod,label='Velocidad eje y',color='blue')
plt.title("Velocidades frente a tiempo")
plt.xlabel("t(s)")
plt.ylabel("v(m/s)")
plt.grid(True)
plt.tight_layout();plt.legend();plt.show()
```

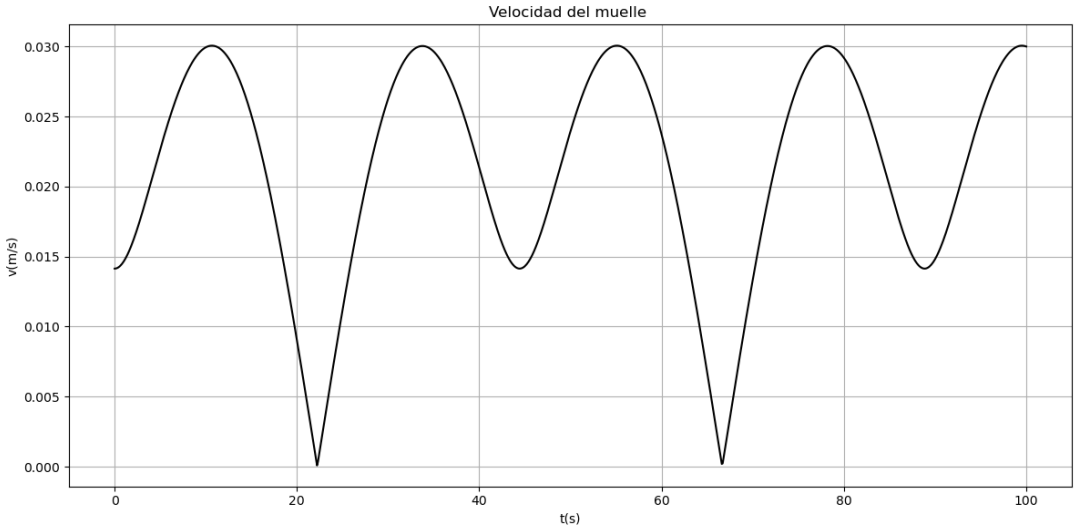


Ahora podemos considerar la velocidad del módulo, y si esta en algún punto es cero quedaría demostrado

```
v=np.zeros(N+1)
```

```
for i in range (N+1):
    v[i]=np.sqrt(vx_eulermod[i]**2+vy_eulermod[i]**2)
```

```
plt.figure(figsize=(12,6))
plt.plot(t,v,color='black')
plt.title("Velocidad del muelle ")
plt.xlabel("t(s)")
plt.ylabel("v(m/s)")
plt.grid(True)
plt.tight_layout();plt.show()
```




```
# Para saber si es 0 el mínimo:
min=np.argmin(v)
print(f"El mínimo tiene un valor de {v[min]:.3g} m/s")
```

```
El mínimo tiene un valor de 8.59e-05 m/s
```

Aunque no sea cero directamente, si tenemos en cuenta los errores del cálculo numérico y de la proximidad al cero del valor, podemos concluir que en dos ocasiones la velocidad es 0.